THESE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par *l'Université Toulouse III - Paul Sabatier*

Discipline ou spécialité : *Informatique*

Présentée et soutenue par *Meghyn GARNER BIENVENU*
Le *7 mai 2009*

Titre : *La génération de conséquences en logique modale*

JURY
BLACKBURN Patrick, DR INRIA Nancy (membre)
GASQUET Olivier PR Université Paul Sabatier (membre)
HERZIG Andreas, DR Université Paul Sabatier (directeur de thèse)
LANG Jérôme, DR Université Paris Dauphine (membre)
MARQUIS Pierre, PR Université d'Artois (rapporteur)
MENGIN Jérôme, MCF Université Paul Sabatier (membre)
ROUSSET Marie-Christine, PR Université de Grenoble (membre)
WOLTER Frank, PR University of Liverpool (rapporteur)

Ecole doctorale : *Mathématiques, Informatique, et Télécommunications*
Unité de recherche : *Institut de Recherche en Informatique de Toulouse*
Directeur(s) de Thèse : *Andreas Herzig*
Rapporteurs : *Pierre Marquis et Frank Wolter*

To Morfar

Acknowledgements

First of all, I would like to thank my thesis advisors Andreas Herzig, Jérome Lang, and Jérome Mengin for all of the advice, support, and encouragement they have provided me over these past few years. I feel truly lucky to have had such excellent thesis advisors, and I sincerely hope that we will find find opportunities to work together again in the future.

I would also like to thank Pierre Marquis and Frank Wolter for kindly accepting to review this thesis, and Patrick Blackburn, Olivier Gasquet, and Marie-Christine Rousset for agreeing to participate in my jury.

A special thanks to Sheila McIlraith, my undergraduate summer project supervisor and first co-author, for helping me take my first steps as a researcher and for always looking out for me as if I were one of her students.

To my friends and colleagues from the LILaC and RPDMP teams at IRIT, thank you for all of the lunches, coffee breaks, and evenings we shared together. I only regret that I was not able to spend more time in Toulouse during my thesis.

To my family, thank you for your continued support over the years, and for flying all the way across the ocean to attend my defense. It meant so much to me to have you all there.

Finally, to Laurent, thank you not only for helping me through the stressful moments, but most of all, for being there to share the happy ones.

Contents

List of Figures

List of Algorithms

Résumé de la thèse

Qu'est-ce que la génération de conséquences ?

La représentation des connaissances est une branche de l'intelligence artificielle qui étudie les différents formalismes permettant de répresenter des informations ainsi que les algorithmes qui permettent d'effectuer différentes tâches de raisonnements sur ces dernières. Une approche courante – celle que l'on adopte dans cette thèse – est d'utiliser des logiques formelles (la logique propositionnelle ou la logique du premier ordre, par exemple) comme langages de représentation des connaissances. Dans cette approche, les informations sont représentées par des formules logiques, et le sens des formules est déterminé par la sémantique de la logique en question.

Lorsque la représentation des connaissances est basée sur une logique formelle, le principal problème lié au concept de raisonnement est celui de la *déduction* : étant données deux formules φ et ψ, l'objectif est de déterminer si ψ est une *conséquence logique* de φ, i.e. si ψ est vérifié chaque fois que φ l'est. Formellement :

$$\text{est-ce que } \varphi \models \psi \,?$$

Nous verrons par la suite que dans certaines situations, une réponse simple de type "oui" ou "non" s'avère insuffisante. On s'intéressera alors à un problème plus général : générer les conséquences logiques d'une formule donnée. Formellement, φ étant donnée :

$$\text{trouver les } \psi \text{ tels que } \varphi \models \psi$$

Cette tâche de raisonnement est communément appelée *génération de conséquences* [Mar00].

Quelles conséquences générer ?

Quand on parle de génération de conséquences, la première question qui se pose est de savoir quelles conséquences on souhaite générer. Nous ne pouvons clairement

pas produire toutes les conséquences logiques d'une formule, car toute formule propositionnelle a une infinité de conséquences. Et même en se restreignant à une seule conséquence par classe d'équivalence, nous produirions toujours beaucoup de conséquences redondantes ou non-pertinentes. Par exemple, si une formule a comme conséquence les formules φ et ψ, alors leur conjonction $\varphi \wedge \psi$ est elle aussi une conséquence de la formule. Or, il semble peu intéressant de générer $\varphi \wedge \psi$ quand nous possédons déjà φ et ψ. De la même façon, si une formule a comme conséquence φ, alors toute formule de la forme $\varphi \vee \psi$ est également une conséquence, mais ces conséquences sont sans grand interêt. Il apparaît donc nécesssaire, avant toute tentative de générer les conséquences d'une formule, de définir le bon sous-ensemble de conséquences "pertinentes" à produire.

Comment peut-on formaliser la notion de "conséquence pertinente" ? En logique propositionnelle, la solution, due à Quine [Qui52, Qui55], est de considérer uniquement *les conséquences clausales les plus fortes* de la formule [1]. Nous appelons ces clauses les *impliqués premiers* de la formule. En ne considérant que des clauses, qui ne contiennent pas de symboles de conjonction, nous éliminons les conséquences du type $\varphi \wedge \psi$, et en ne gardant que les conséquences clausales les plus fortes, nous éliminons les conséquences plus faibles de type $\varphi \vee \psi$, où φ est une conséquence. Comme chaque formule propositionnelle est équivalente à une conjonction de clauses, et chaque clause qui est impliquée par une formule φ est impliquée par un des impliqués premiers de φ, les impliqués premiers donnent une représentation complète et succincte de l'ensemble des conséquences logiques d'une formule.

La génération de conséquences, à l'envers

Dans certaines situations, on s'intéresse non pas à l'ensemble des conséquences logiques d'une formule φ, mais plutôt à l'ensemble des "causes" de φ, c'est-à-dire l'ensemble des formules qui ont φ pour conséquence logique. Autrement dit, nous voulons faire de la génération de conséquences à l'envers. Formellement, étant donnée φ :

$$\text{trouver les } \psi \text{ tels que } \psi \models \varphi$$

Comme pour la génération de conséquences « standard », il faut décider du type de formules que l'on souhaite produire : parmi toutes les formules ψ telles que $\psi \models \varphi$, quelles sont les formules intéressantes à générer ?

1. Nous rappelons qu'en logique propositionnelle un littéral est soit une variable propositionnelle soit la négation d'une variable propositionnelle, et qu'une clause est une disjonction de littéraux, e.g. $a \vee \neg b \vee \neg c$.

Comme nous faisons le contraire de la génération de conséquences, ce qu'il nous faut est l'opposé d'un impliqué premier ! Au lieu de considérer les clauses, nous utilisons la notion duale de termes [2], et au lieu de prendre les formules les plus fortes, nous prenons les plus faibles. La notion que nous obtenons ainsi est connue sous le nom d'*implicant premier*.

Comme on peut l'imaginer, les notions d'impliqué et d'implicant premier sont fortement liées. En effet, chacune de ces deux notions peut être définie en fonction de l'autre : les impliqués premiers de φ sont toutes les clauses dont la négation est équivalente à l'un des implicants premiers de $\neg\varphi$, et les implicants premiers de φ sont tous les termes dont la négation est équivalente à l'un des impliqués premiers de $\neg\varphi$. Grace à cette dualité, tous les résultats que nous obtiendrons sur les impliqués premiers pourront être transférés aux implicants premiers, et vice-versa.

Impliqués et implicants premiers : quelle utilité ?

Les impliqués et implicants premiers ont été utilisés dès les années 1950 dans le domaine de la conception de circuits électroniques. En effet, trouver un circuit de hauteur 2 représentant une formule φ donnée et possédant le moins de portes possibles revient à trouver la représentation la plus compacte de φ comme disjonction d'implicants premiers ou comme conjonction d'impliqués premiers (voir Chapitre 4 de [BV04]). Les tableaux de Karnaugh, un classique des cours d'informatique de Licence, ne sont rien de plus qu'une méthode visuelle pour trouver une telle représentation dans le cas de formules à 2 ou 3 variables. De même, le célèbre algorithme de minimisation de Quine-McCluskey [McC56] commence par calculer l'intégralité des implicants/impliqués premiers d'une formule, pour ensuite en extraire une couverture de coût minimal. Pour des formules à grand nombre de variables, des algorithmes heuristiques comme Espresso [BSVMH84] permettent de trouver une couverture de faible coût, sans toutefois en assurer l'optimalité.

A partir la fin des années 1980, les impliqués et implicants premiers ont fait leur apparition dans le domaine de l'intelligence artificielle. Depuis, ils ont été appliqués à de nombreuses problématiques, comme le raisonnement distribué [ACG$^+$06], la révision des croyances (cf. [Bit07], [Pag06], [BHQ08]), le raisonnement non-monotone (cf. [Prz89]), et l'étude de la pertinence (cf. [Lak95], [LLM03]). Mais les champs d'application les plus importants sont sans doute la compilation de connaissances et le raisonnement abductif. Dans la suite de cette section, nous étudions en détail le rôle de la génération de conséquences dans ces deux domaines.

2. Un terme est une conjonction de littéraux.

Compilation de connaissances

Lorsque l'on base la représentation des connaissances sur la logique, une difficulté se présente immédiatement : les tâches de raisonnement ont une haute complextié algorithmique. En effet, même pour la logique propositionnelle, qui est parmi les moins expressives des logiques communément utilisées en représentation des connaissances, le problème de la déduction est co-NP-complet [3]. Il y a donc peu d'espoir de trouver des algorithmes de raisonnement qui terminent en un temps acceptable sur toutes les entrées.

La compilation de connaissances (cf. [CD97], [DM02]) est une technique générale pour faire face à la complexité élévée du raisonnement. Elle comporte deux phases : une phase préliminaire « hors ligne » dans laquelle la base de connaissances initiale (pour nous, une base est simplement une formule dans une logique donnée) est remplacée par une base équivalente dont la structure permettra, par la suite, un raisonnment efficace. S'en suit la phase « en ligne » dans laquelle nous effectuons des tâches de raisonnement sur la nouvelle base de connaissances. La phase préliminaire peut être difficile et coûteuse, mais l'idée est que ce coût initial sera compensé par les économies réalisées sur le raisonnement effectué pendant la deuxième phase.

Il existe un certain nombre de méthodes différentes pour compiler les formules de la logique propositionnelle, mais l'une des méthodes les plus connues est de représenter des formules par la conjonction de leurs impliqués premiers :

$$\varphi \quad \longmapsto \quad \pi_1 \wedge ... \wedge \pi_n$$

Les formules propositionnelles sous cette forme ont de bonnes propriétés calculatoires. En particulier, il est possible de tester en temps polynomial si une formule sous cette forme implique une formule en forme normale conjonctive (FNC). Afin de comprendre pourquoi, remarquons que ce dernier problème

$$\pi_1 \wedge ... \wedge \pi_n \models \lambda_1 \wedge ... \wedge \lambda_m$$

se reduit à vérifier que pour chacune des clauses λ_i nous avons

$$\pi_1 \wedge ... \wedge \pi_n \models \lambda_i$$

Si les formules π_j étaient des clauses quelconques, ce dernier problème serait très difficile (co-NP-dur, pour être précis). Mais comme ce sont des impliqués premiers, nous pouvons profiter du fait que chaque clause impliquée par une formule doit l'être par l'un des impliqués premiers de la formule. En conséquence, nous avons simplement besoin de tester si l'une des formules π_j implique λ_i :

$$\text{pour chaque } \lambda_i, \text{ tester s'il existe } \pi_j \text{ tel que } \pi_j \models \lambda_i$$

3. Consulter l'Appendice A pour les définitions des classes de complexité.

Enfin, nous remarquons que comme π_j et λ_i sont des clauses, tester si $\pi_j \models \lambda_i$ est aussi simple que de vérifier que chaque littéral dans la clause π_j est aussi dans λ_i :

$$\text{les littéraux de } \pi_j \text{ sont-ils tous des littéraux de } \lambda_i \,?$$

Nous avons décrit un algorithme simple et efficace pour tester si une formule représentée par ses impliqués premiers implique une formule donnée en FNC.

Représenter les formules comme conjonction de leurs impliqués premiers permet également deux types de transformations importants : *le conditonnement*, où l'on assigne une valeur de vérité à l'une des variables propositionnelles d'une formule, et *l'interpolation uniforme*, où l'on projette une formule sur une signature donnée [4]. Pour comprendre pourquoi cette dernière tâche est facile (au sens algorithmique), remarquons que les impliqués premiers de la projection d'une formule sur une signature sont précisément les impliqués premiers de la formule qui ne contiennent que des variables appartenant à la signature. Cela veut dire que si une formule est représentée comme conjonction de ses impliqués premiers, l'interpolation uniforme est aussi simple que d'enlever de la conjonction tous les impliqués premiers qui contiennent des variables n'appartenant pas à la signature.

Raisonnement abductif

L'abduction est un type de raisonnement dont le but est de produire des explications possibles pour une observation donnée. Ce mode de raisonnement est central dans plusieurs domaines de l'intelligence artificielle, par exemple, le diagnostic, la planification, la compréhension du langage naturel, et la vision par ordinateur (se reporter à [EG95] pour les références). Du point du vue de la logique, un problème d'abduction consiste en une observation (ce que nous voulons expliquer) et un ensemble de connaissances, tous les deux représentés par des formules logiques. L'objectif est de trouver une *explication*, c'est-à-dire une formule qui implique l'observation (o) étant données les connaissances présupposées (t) :

$$\text{trouver les } e \text{ tels que } t \wedge e \models o$$

Bien sûr, le nombre d'explications possibles peut être très important ; nous avons donc une nouvelle fois besoin d'un sous-ensemble d'explications "pertinentes". Si nous réécrivons la tâche d'abduction comme ceci

$$\text{trouver les } e \text{ tels que } e \models \neg t \vee o$$

4. En logique propositonnelle, une signature est un ensemble de variables propositionnelles.

alors la réponse est évidente : nous devons utiliser les implicants premiers ! L'ensemble des explications pertinentes est alors simplement l'ensemble des implicants premiers de $\neg t \vee o$.

En réalité, un peu de prudence s'impose. Parmi les implicants premiers de $\neg t \vee o$ figurent les implicants premiers de $\neg t$, ce qui signifie que certaines des explications que nous générons peuvent être en contradiction avec la base de connaissances t. Ceci est clairement indésirable. Pour éliminer ces explications insatisfaisantes, nous exigeons de plus que les explications soient compatibles avec la base de connaissances. Avec cette restriction supplémentaire, on obtient une version plus sophistiquée du problème d'abduction :

$$\text{trouver les } e \text{ tels que } t \wedge e \models o \text{ et } t \wedge e \not\models \bot$$

qui correspond à la tâche suivante

$$\text{trouver les } e \text{ tels que } e \models \neg t \vee o \text{ et } e \not\models \neg t$$

Ce que nous cherchons vraiment est donc l'ensemble des implicants premiers de $\neg t \vee o$ qui n'impliquent pas $\neg t$. Cette variante de la notion d'implicant premier a été étudiée dans la littérature (cf. [Ino92], [del99] et discussion dans [Mar00]).

Une autre restriction couramment imposée à l'ensemble des explications est de demander que celles-ci soient construites à partir d'une signature donnée (cf. [EG95], [SL96]). Cela veut dire que nous souhaitons produire des implicants premiers qui ne contiennent que des variables propositionnelles de la signature. Cette variante de la notion d'implicant premier a été étudiée longuement dans la littérature, et il existe plusieurs algorithmes de génération de conséquences qui peuvent produire des implicants premiers de ce type (cf. [Ino92], [del99], [SdV01], et discussion dans [Mar00]).

Au-delà de la logique classique

Pour de nombreuses applications en intelligence artificielle, la puissance expressive de la logique propositionnelle s'avère insuffisante. La logique du premier ordre offre une très grande expressivité, mais au prix de l'indécidabilité. Les logiques modales et les logiques de description sont deux familles de logiques qui proposent un bon compromis entre expressivité et complexité, car elles sont en général plus expressives que la logique propositionnelle mais possèdent de meilleures propriétés calculatoires que la logique du premier ordre. Ceci explique la tendance croissante à utiliser ces logiques pour la représentation des connaissances.

Une limitation de la recherche actuelle sur la génération de conséquences est qu'elle se focalise quasi-exclusivement sur la logique propositionnelle et la logique du premier ordre. A notre connaissance, la génération de conséquences pour les logiques modales ou les logiques de description n'a jamais été étudiée. Cette lacune s'explique peut-être par le fait que la plupart des logiques modales et des logiques de description correspondent à des fragments de la logique du premier ordre. Il peut donc sembler inutile d'étudier la génération de conséquences pour ces logiques, car il est possible de traduire les formules de ces logiques en formules du premier ordre, puis d'appliquer les résultats et algorithmes déjà proposés pour la logique du premier ordre.

Le défaut de cet argument est que les impliqués et implicants premiers ne se comportent pas aussi bien en logique du premier ordre qu'en logique propositionnelle. En effet, nous perdons en logique du premier ordre quelques propriétés clés comme la finitude (les formules peuvent avoir une infinité d'impliqués premiers distincts) et l'équivalence (une formule n'est pas nécéssairement équivalente à l'ensemble de ses impliqués premiers) [Mar91b, Mar91a]. Et comme par ailleurs les logiques modales ou de description possèdent souvent de bien meilleures propriétés calculatoires que la logique du premier ordre, on peut raisonnablement espérer obtenir de meilleurs résultats en faisant la génération de conséquences directement dans ces logiques, plutôt que de passer par la logique du premier ordre.

C'est pourquoi dans cette thèse nous proposons une étude de la génération de conséquences en logique modale, et plus précisement, dans la logique modale $\mathcal{K}_n$. Nous avons choisi la logique $\mathcal{K}_n$ pour deux raisons : d'une part il s'agit de la logique modale prototypique, et d'autre part elle possède de forts liens avec la logique de description $\mathcal{ALC}$.

La question de savoir comment définir de façon appropriée les impliqués et implicants premiers dans la logique $\mathcal{K}_n$ est clairement intéressante d'un point de vue théorique. Nous soutenons de plus qu'une solution satisfaisante serait prometteuse en termes d'applications. Passons brièvement en revue deux domaines d'application possibles.

Un premier domaine d'application potentiel est le raisonnement abductif dans $\mathcal{K}_n$. Comme on l'a rappelé ci-dessus, l'une des questions fondamentales du raisonnement abductif est la sélection d'un sous-ensemble d'explications intéressantes. Cette question se pose avec encore plus de force pour les logiques comme $\mathcal{K}_n$, pour lesquelles il peut y avoir une infinité de formules non-équivalentes (le nombre d'explications distinctes pour un problème d'abduction peut donc être infini), rendant *de facto* impossible la génération de toutes les explications. Comme les implicants

premiers constituent la notion-clé permettant de caractériser les explications pertinentes en logique propositionnelle, il semble qu'un bon point de départ de l'étude du raisonnement abductif dans $\mathcal{K}_n$ serait de définir une notion d'impliqué premier pour cette logique. Nous allons dans ce qui suit proposer plusieurs définitions possibles, et comparer leurs propriétés respectives.

L'étude des impliqués premiers dans la logique $\mathcal{K}_n$ pourrait également se montrer utile pour le développement de méthodes de compilation pour cette logique. Actuellement, la quasi-totalité des travaux sur la compilation de connaissances se concentre sur la logique propositionnelle, bien que cette technique pourrait être étendue aux logiques modales et de description, pour lesquelles les tâches de raisonnement sont encore plus complexes qu'en logique propositionnelle. Ici encore, le rôle crucial de la notion d'impliqué premier en compilation de connaissances en logique propositionnelle suggère d'étendre cette notion à des logiques plus riches comme $\mathcal{K}_n$.

Organisation de la thèse

Cette thèse constitue une exploration de la génération de conséquences dans la logique multi-modale $\mathcal{K}_n$. Les principales questions que nous allons traiter sont les suivantes :

- Comment peut-on définir de façon appropriée les notions d'impliqué et implicant premier dans $\mathcal{K}_n$? Quelles propriétés conservent ces notions par rapport à la logique propositionnelle ?
- Comment peut-on générer les impliqués premiers d'une formule de $\mathcal{K}_n$?
- Comment peut-on tester si une formule est un impliqué premier ? Quelle est la complexité de cette tâche ?
- Combien d'impliqués premiers une formule peut-elle avoir ? Quelle est leur taille ?
- Comment utiliser les impliqués premiers pour compiler des formules dans $\mathcal{K}_n$?

Nous présentons maintenant un bref aperçu des différents chapitres qui composent cette thèse.

Chapitre 2. Ce chapitre constitue une introduction à la logique $\mathcal{K}_n$. Les sujets abordés comprennent : la syntaxe et la sémantique de $\mathcal{K}_n$, la terminologie et les notations, les propriétés de la conséquence logique, des transformations sur les formules dans $\mathcal{K}_n$, les principales tâches de raisonnement et leur complexité, et les relations entre la logique $\mathcal{K}_n$, la logique du premier ordre, et les logiques de des-

cription.

Chapitre 3. Dans ce chapitre, nous abordons la question de savoir comment les notions d'impliqué et d'implicant premier peuvent être définies de façon appropriée dans la logique $\mathcal{K}_n$. Comme les impliqués et implicants premiers sont définis en logique propositionnelle au moyen des notions syntaxiques de clause et de terme, qui ne sont pas des notions standard dans $\mathcal{K}_n$, nous commençons le chapitre par une étude de plusieurs définitions possibles de clauses et de termes dans $\mathcal{K}_n$. Les différentes définitions sont évaluées à l'aune de leurs propriétés syntaxiques, sémantiques, et calculatoires. Nous commençons par définir un ensemble "idéal" de propriétés que l'on aimerait voire satisfaites par les clauses et les termes. Nous montrons qu'hélas, aucune de nos définitions de clause et de terme ne satisfait toutes ces propriétés (nous montrerons en effet qu'*aucune* définition possible ne les satisfait), mais deux de nos définitions s'en approchent raisonnablement. Dans la seconde partie du chapitre, nous examinons de nouveau les définitions proposées en fonction des notions d'impliqués et d'implicants premiers qu'elles induisent. Nous montrons alors qu'une seule de nos définitions candidates pour les notions de terme et clause induit des notions satisfaisantes d'impliqués et d'implicants premiers.

Chapitre 4. Ce chapitre étudie les propriétés calculatoires de la notion d'impliqué premier que nous avons séléctionnée dans le chapitre 3. Dans la première moitié du chapitre, nous proposons un algorithme correct et complet, **GenPI**, pour générer l'ensemble des impliqués premiers d'une formule de $\mathcal{K}_n$. Notre algorithme fonctionne par décomposition : d'abord, nous écrivons la formule d'origine comme une disjonction de formules plus simples, puis nous calculons les impliqués premiers de ces dernières, et enfin nous utilisons ces impliqués premiers pour calculer les impliqués premiers de la formule d'origine.

Une analyse de la structure des impliqués premiers construits par notre algorithme nous permet de donner des bornes supérieures sur la taille et le nombre des impliqués premiers. Précisement, nous prouvons que chaque impliqué premier d'une formule φ est équivalent à une clause qui est au plus exponentiellement plus grande que φ et qu'une formule ne peut possèder dans le pire des cas qu'un nombre doublement exponentiel d'impliqués premiers distincts. Nous démontrons que ces bornes sont optimales en donnant des bornes inférieures correspondantes, et nous prouvons que ces bornes restent valables même pour des notions d'impliqués premiers beaucoup moins expressives.

La deuxième moitié du chapitre concerne la reconnaissance des impliqués premiers, qui est le problème de savoir si une clause est un impliqué premier d'une

formule donnée. Bien que cette question soit intéressante en soi, notre motivation principale est d'améliorer la complexité de notre algorithme de génération **GenPI**, qui utilise une méthode inefficace pour vérifier si une clause candidate est bien un impliqué premier. Nous proposons un algortihme correct et complet **TestPI** pour la reconnaissance des impliqués premiers, et nous montrons qu'il s'effectue en espace polynomial. Cela nous permet de prouver que le problème de reconnaissance est PSPACE-complet, et donc de même complexité que la satisfiabilité et la déduction dans $\mathcal{K}_n$.

Chapitre 5. Nous avons remarqué plus haut que certaines applications (comme l'abduction) peuvent nécessiter des variantes plus raffinées de la notion d'impliqué premier. C'est pourquoi dans le chapitre 5 nous examinons deux variantes de notre notion d'impliqué premier : les *nouveaux impliqués premiers*, qui nous permettent d'isoler les nouveaux faits que l'on peut déduire après l'ajout d'une information, et *les impliqués premiers sur une signature*, qui nous permettent de caractériser les conséquences d'une formule construites à partir d'une signature donnée. Nous étudions les propriétés de ces deux notions, en s'appuyant sur les résultats des chapitre précédents. Nous montrons en particulier que les impliqués premiers sur une signature n'ont pas d'aussi bonnes propriétés calculatoires que les impliqués premiers standards.

Chapitre 6. Dans ce chapitre, nous nous interrogeons sur la possibilité d'utiliser notre notion d'impliqué premier afin de faire de la compilation de connaissances dans $\mathcal{K}_n$, comme cela se fait en logique propositionnelle. En début de chapitre, nous expliquons pourquoi la façon la plus simple de définir une forme normale à partir de nos impliqués premiers n'est pas satisfaisante. Ceci nous amène à proposer notre propre définition, plus sophistiquée. Nous étudions les propriétés de notre forme normale, montrant en particulier qu'elle permet un test d'implication simple par comparaison syntaxique (qui est assez proche de le procédure décrite plus tôt dans le chapitre pour la logique propositionnelle).

Nous montrons aussi que l'interpolation uniforme est facile (au sens calculatoire) pour les formules de $\mathcal{K}_n$ mises sous forme normale. Nous étudions ensuite la complexité de la mise sous forme normale des formules de $\mathcal{K}_n$, et proposons un algorithme pour effectuer cette tâche. Nous concluons le chapitre par une comparaison de notre forme normale aux autres formes normales pour $\mathcal{K}_n$ proposées dans la littérature.

Chapitre 7. Ce chapitre résume les principales contributions de la thèse, et propose quelques pistes intéressantes pour des recherches futures.

1

Introduction

What is consequence finding?

Knowledge representation is a subfield of artificial intelligence which is concerned with the study of formalisms for representing different kinds of information and the development of procedures for performing reasoning on these representations. Many knowledge representation formalisms exist, but one popular approach, and the one we adopt in this thesis, is to utilize formal logics (propositional logic, first-order logic, etc.) as knowledge representation languages. According to this approach, information is represented using logical formulae, and the meaning of formulae is determined by the semantics of the logic in question.

The major reasoning task in logic-based knowledge representation is that of *deduction*: given two formulae, let's call them φ and ψ, our job is to determine whether ψ is a *logical consequence* of φ, i.e. whether the truth of φ guarantees the truth of ψ. Symbolically,

$$\text{does } \varphi \models \psi?$$

As we shall see later in the section, there are circumstances in which such a simple "yes" or "no" answer proves insufficient. Instead, what we are interested in is the more general problem of generating logical consequences of a particular formula:

$$\text{find } \psi \text{ such that } \varphi \models \psi$$

This reasoning task is commonly known as *consequence finding* [Mar00].

But which consequences should we generate?

One of the first questions that presents itself when we talk about consequence finding is which consequences do we generate? We obviously cannot generate *all* of the consequences of a formula, because even the simplest propositional formula has infinitely many consequences. Even if we restrict ourselves to one consequence per equivalence class, we still produce a lot of clearly irrelevant or redundant consequences. Indeed, if a formula has both φ and ψ as consequences, then the formula $\varphi \wedge \psi$ is also a consequence, but it seems entirely superfluous once we have φ and ψ. Likewise, if a formula has a consequence φ, then every formula of the form $\varphi \vee \psi$ is also a consequence, but these consequences don't seem to hold much interest. What we need is a way of focusing in on *a relevant subset of consequences.*

How can we formalize the notion of a relevant or interesting consequence? In propositional logic, the solution, due to Quine [Qui52, Qui55], is to consider only the logically strongest clauses [1] which are consequences of the formula. We call these clauses the formula's *prime implicates*. By focusing on clauses, which do not contain any conjunction symbols, we avoid redundant consequences of the type $\varphi \wedge \psi$, and by only considering the logically strongest clausal consequences, we eliminate weaker, irrelevant consequences of the type $\varphi \vee \psi$. As every propositional formula can be rewritten as a conjunction of clauses, and every clausal consequence of a formula is entailed by some prime implicate of the formula, prime implicates provide a complete yet compact representation of a formula's consequences.

Consequence finding, in reverse

In some circumstances, we may be interested not in the logical consequences of a given formula, but rather the formulae which have this formula as a consequence. Basically, we want to do consequence finding in reverse:

$$\text{find } \psi \text{ such that } \psi \models \varphi$$

Just as for standard consequence finding, a key issue is selecting the right set of formulae to generate: of the many ψ which satisfy $\psi \models \varphi$, which ones should we choose?

Well, since we are doing the opposite of consequence finding, what we need is the opposite of a prime implicate! Instead of clauses, we can use the dual notion of terms [2], and instead of taking the logically strongest formulae, we take the logically weakest. The resulting notion is known as a *prime implicant*.

1. We recall that in propositional logic a literal is a propositional variable or the negation of a propositional variable, and a clause is a disjunction of literals, e.g. $a \vee \neg b \vee \neg c$.

2. Terms are conjunctions of propositional literals.

As one might expect, prime implicates and prime implicants are very closely related. Indeed, each of these notions can be defined in terms of the other: the prime implicates of φ are just the clauses which are equivalent to the negation of a prime implicant of $\neg\varphi$, and the prime implicants of φ are precisely those terms whose negations are equivalent to prime implicates of $\neg\varphi$. This means that all of the results concerning prime implicates can be transferred to prime implicants, and vice-versa.

Prime implicates and prime implicants: what are they good for?

Prime implicates and prime implicants have been used since the fifties in the field of digital circuit synthesis: the design of minimal-cost two-level circuits comes down to finding the shortest way of representing a propositional formula as either a disjunction of a subset of its prime implicants or a conjunction of some of its prime implicates (cf. Chapter 4 of [BV04]). Karnaugh maps, a staple of undergraduate computer science courses, are really nothing more than a visual tool for isolating covering sets of prime implicants/implicates, and the famous Quine-McCluskey minimization algorithm [McC56] works by first generating the entire set of prime implicants/implicates, then computing the covering subsets with minimal cost. For circuits with large numbers of variables, heuristic methods, like Espresso [BSVMH84], allow one to produce good but not necessarily optimal prime implicate/implicant covers without the computation of the entire set of prime implicates/implicants.

Starting from the late eighties, prime implicates and prime implicants began to appear in the artificial intelligence literature. Since then, these notions have been utilized for a number of different AI problems, such as distributed reasoning [ACG$^+$06], belief revision (cf. [Bit07], [Pag06], [BHQ08]), non-monotonic reasoning (cf. [Prz89]), and characterizations of relevance (cf. [Lak95], [LLM03]). Probably the most important domains of application, however, are knowledge compilation and abductive reasoning. In the remainder of this section, we present a detailed look at the role of consequence finding in these two areas.

Knowledge compilation

One major obstacle for logic-based knowledge representation is the high computational complexity of reasoning. Indeed, even for propositional logic, which is among the least expressive knowledge representation languages, the basic reasoning task of deduction is co-NP-complete [3]. This means that there is little hope of

3. Refer to Appendix A for the definitions of this and other complexity classes.

finding reasoning algorithms which terminate in a reasonable amount of time on all inputs.

Knowledge compilation (cf. [CD97], [DM02]) is a general technique for coping with the intractability of reasoning. It consists of two phases: a preliminary off-line phase in which we replace the original knowledge base (for us, this is just a formula in some logic) by an equivalent knowledge base which admits efficient reasoning, followed by a second online phase in which we perform reasoning tasks on the compiled knowledge base. The off-line phase may prove difficult and costly, but the idea is that this initial cost will be offset by the computational savings on the reasoning done during the online phase.

There exist a number of different methods for compiling propositional formulae, but one of the better-known approaches is to use prime implicate normal form [4], in which a formula is represented as the conjunction of its prime implicates:

$$\varphi \quad \longmapsto \quad \pi_1 \wedge ... \wedge \pi_n$$

Propositional formulae in prime implicate normal form have many nice computational properties. In particular, it is possible to test in polynomial time whether a formula in prime implicate normal form entails a formula in conjunctive normal form (CNF). To see why, we first remark that this problem

$$\pi_1 \wedge ... \wedge \pi_n \models \lambda_1 \wedge ... \wedge \lambda_m$$

can be reduced to testing whether for each of the clauses λ_i we have

$$\pi_1 \wedge ... \wedge \pi_n \models \lambda_i$$

Now if the conjuncts π_j were arbitrary clauses, then the latter problem would be very difficult (co-NP-hard, to be precise). But because we are dealing with prime implicates, we can take advantage of the fact that every clause implied by a formula must be implied by one of the formula's prime implicates. This means that we just need to find a *single* conjunct π_j which implies λ_i:

$$\text{for each } \lambda_i, \text{ check whether there is some } \pi_j \text{ such that } \pi_j \models \lambda_i$$

Finally, we remark that since the π_j and λ_i are all clauses, deciding whether $\pi_j \models \lambda_i$ is easy since we just need to test whether each of the literals appearing in π_j also appears in λ_i:

$$\text{is each disjunct of } \pi_j \text{ also a disjunct of } \lambda_i?$$

4. Prime implicant normal form also exists, but is a bit less common. It offers many of the same advantages as prime implicate normal form, the exception being that the uniform interpolation transformation is not tractable [DM02].

We have thus outlined a simple and efficient procedure for determining whether a formula in prime implicate normal form implies a formula in CNF. Notice that this procedure can also be used to decide entailment or equivalence between two formulae in prime implicate normal form in polynomial time.

Prime implicate normal form also supports two important transformations: *conditioning*, in which we assign a truth value to one of a formula's propositional variables, and *uniform interpolation* (or forgetting), in which we approximate a formula over a given signature [5]. To see why the latter task is tractable, we remark that the prime implicates of the approximation of a formula over a signature are precisely those prime implicates of the original formula which do not contain any propositional variables outside the signature. This means that for formulae in prime implicate normal form, uniform interpolation is as simple as removing those conjuncts which contain one of the unwanted propositional variables.

Abductive reasoning

Abduction is a form of reasoning that is used to generate explanations for observations. It has been applied to a number of different areas in artificial intelligence, e.g. diagnosis, planning, natural language understanding, and computer vision (refer to [EG95] for references). In logic-based approaches to abduction, an abduction problem typically consists of an observation (what we want to explain) and some background knowledge, both of which are represented by logical formulae. The objective is to find an *explanation*, that is, a formula which logically entails the observation (o) when taken together with the background theory (t):

$$\text{find } e \text{ such that } t \wedge e \models o$$

Of course, the number of possible explanations might be very large, so we need a way of characterizing the interesting explanations. If we rewrite the abduction task in terms of reverse consequence finding as follows

$$\text{find } e \text{ such that } e \models \neg t \vee o$$

then the answer becomes obvious: we should use prime implicants! Thus, we can define the set of interesting explanations to be the prime implicants of $\neg t \vee o$.

Well, actually we need to be a bit more careful. Among the prime implicants of $\neg t \vee o$ are the prime implicants of $\neg t$, which means that some of the explanations we generate may be in contradiction with the background knowledge. This is clearly undesirable, so in order to eliminate these unsatisfactory explanations, we generally

5. In propositional logic, a signature is a just a set of propositional variables.

place an additional requirement on explanations, namely that they be consistent with the background theory. This yields the following more sophisticated abduction problem:

$$\text{find } e \text{ such that } t \wedge e \models o \text{ and } t \wedge e \not\models \bot$$

which corresponds, in consequence finding terms, to the following task

$$\text{find } e \text{ such that } e \models \neg t \vee o \text{ and } e \not\models \neg t$$

So what we are after are those prime implicants of $\neg t \vee o$ which do not imply $\neg t$. This variant on the basic notion of prime implicant has been investigated in the consequence finding literature (cf. [Ino92], [del99] and discussion in [Mar00]).

Another common restriction on explanations is to require that they are built from a specified signature (cf. [EG95], [SL96]). In consequence-finding terms, this means that we want to look for prime implicants which only contain propositional variables belonging to the given signature. This more refined notion of prime implicant has been studied extensively in the literature, and many consequence-finding algorithms exist for producing prime implicants of this type (cf. [Ino92], [del99], [SdV01], and discussion in [Mar00]).

Going beyond classical logic

For many applications in artificial intelligence, the expressive power of propositional logic proves insufficient. First-order logic provides a much greater level of expressivity, but at the price of undecidability. Modal and description logics are two families of logics which offer an interesting trade-off between expressivity and complexity, as they are generally more expressive than propositional logic yet are better-behaved computationally than first-order logic. This explains the growing trend towards using such languages for knowledge representation.

One limitation of current research in consequence finding is that it is focused almost exclusively on classical propositional and first-order logic (with an emphasis on the former). To our knowledge, there has not been any research concerning consequence finding for modal and description logics. Perhaps one explanation for this is that most modal and description logics correspond to fragments of first-order logic. Thus, one might argue that it is unnecessary to study consequence finding for these logics, since we can just map our formulae to first-order logic and do consequence finding there.

The problem with this argument is that prime implicates and prime implicants do not behave as nicely in first-order logic as in propositional logic. Indeed, we lose some key properties like finiteness (first-order logic formulae can have infinitely

many distinct prime implicates) and equivalence (a first-order formula is not necessarily equivalent to its set of prime implicates) [Mar91b, Mar91a]. Given that modal and description logics have better computational properties than first-order logic, there is reason to believe that we might have better luck doing consequence finding directly in these logics, without passing by first-order logic.

This is why in this thesis we propose to study consequence finding in modal logic, and more specifically, in the modal logic $\mathcal{K}_n$. We decided to study $\mathcal{K}_n$ because it is the prototypical modal logic, and also because of its close relationship with the well-known description logic $\mathcal{ALC}$. Indeed, while the results in this thesis are presented in terms of $\mathcal{K}_n$ formulae, all of our results hold equally well for $\mathcal{ALC}$ concept expressions.

The question of how the notions of prime implicates and prime implicants can be suitably defined for the logic $\mathcal{K}_n$ is clearly of interest from a theoretical point of view. We argue, however, that this question is also practically relevant. To support this claim, we briefly discuss two application areas in which the study of prime implicates and prime implicants in $\mathcal{K}_n$ might prove useful.

One potential domain of application is abductive reasoning in $\mathcal{K}_n$. As noted above, one of the key foundational issues in abductive reasoning is the selection of an interesting subset of explanations. This issue is especially crucial for logics like $\mathcal{K}_n$ which allow for an infinite number of non-equivalent formulae, since this means that the number of non-equivalent explanations for an abduction problem is not just large but in fact infinite, making it simply impossible to enumerate the entire set of explanations. As prime implicants are a widely-accepted means of characterizing relevant explanations in propositional logic, a reasonable starting point for research into abductive reasoning in the logic $\mathcal{K}_n$ is the study of different possible definitions of prime implicant in $\mathcal{K}_n$ and their properties.

The investigation of prime implicates in $\mathcal{K}_n$ is also relevant to the development of knowledge compilation procedures for $\mathcal{K}_n$. Currently, most work on knowledge compilation is restricted to propositional logic, even though this technique could prove highly relevant for modal and description logics, which generally suffer from an even higher computational complexity than propositional logic. As prime implicates are one of the better-known mechanisms for compiling formulae in propositional logic, it certainly makes sense to investigate whether this approach to knowledge compilation can be fruitfully extended to logics like $\mathcal{K}_n$.

Organization of this thesis

This thesis constitutes an exploration of consequence finding in the basic modal logic $\mathcal{K}_n$. The main questions that we will be addressing are the following:

- How can prime implicates and prime implicants be appropriately defined in the logic $\mathcal{K}_n$? What are the properties of the resulting notions?
- How can one generate the prime implicates of formulae in $\mathcal{K}_n$?
- How can one test whether a formula is indeed a prime implicate? What is the complexity of this task?
- How many prime implicates can a $\mathcal{K}_n$-formula have? How large can these prime implicates be?
- How can prime implicates be used for compiling $\mathcal{K}_n$-formulae?

We now present a brief overview of the different chapters of this thesis.

Chapter 2. This chapter provides the necessary background material for later chapters. Topics covered include: syntax and semantics of $\mathcal{K}_n$, terminology and notation, properties of logical consequence, transformations on $\mathcal{K}_n$ formulae, principal reasoning tasks and their complexities, and the relationship of $\mathcal{K}_n$ to first-order logic and description logics.

Chapter 3. In this chapter, we address the question of how the notions of prime implicates and prime implicants can be appropriately lifted from propositional logic to $\mathcal{K}_n$. As prime implicates and prime implicants are defined in terms of the notions of clauses and terms, which are not standard notions in $\mathcal{K}_n$, we begin the chapter by considering a number of potential definitions of clauses and terms for $\mathcal{K}_n$. The different definitions are evaluated with respect to a set of syntactic, semantic, and complexity-theoretic properties characteristic of the propositional definition. None of the definitions satisfies all of these properties (indeed we show this to be impossible), but two of the definitions come reasonably close. In the second half of the chapter, we take a second look at the candidate definitions, this time evaluating them with respect to the properties of the notions of prime implicates and prime implicants that they induce. We show that only one of the candidate definitions yields a satisfactory notion of prime implicates and prime implicants.

Chapter 4. This chapter investigates the computational properties of our selected definition of prime implicate. In the first half of the chapter, we propose a sound and complete algorithm **GenPI** for generating prime implicates. Our algorithm adopts a decomposition-style approach: first, we rewrite the original formula as

a disjunction of simpler formulae, then we compute the prime implicates of these simpler formulae, and finally, we use the prime implicates of the simpler formulae to help us compute the prime implicates of the original formula.

An analysis of the structure of the prime implicates constructed by our algorithm allows us to place upper bounds on the size and number of prime implicates. Specifically, we demonstrate that every prime implicate of a formula is equivalent to a clause which is no more than single-exponentially larger than the formula, and that a formula can possess no more than double-exponentially many prime implicates modulo equivalence. We prove these upper bounds optimal by providing matching lower bounds, and then we go further and show that the lower bounds hold even for much less expressive notions of prime implicates.

The focus of the second half of the chapter is on prime implicate recognition, which is the problem of deciding whether a given clause is a prime implicate of a formula. While this problem is interesting in and of itself, an additional motivation for studying this task is to improve our generation algorithm **GenPI**, which utilizes a very inefficient method for verifying whether a candidate clause is indeed a prime implicate. We propose a sound and complete procedure **TestPI** for recognizing prime implicates, and we show that it runs in polynomial space. This allows us to prove that the prime implicate recognition task is PSPACE-complete, and hence of the same complexity as standard reasoning tasks in $\mathcal{K}_n$.

Chapter 5. We saw earlier in the chapter that some applications (like abduction) can require more refined notions of prime implicates/implicants. This is why in Chapter 5 we study two variants on our notion of prime implicate: new prime implicates, which allow us to isolate the novel facts which can be derived upon arrival of new information, and signature-bounded prime implicates, which allow one to characterize the consequences of a formula which are built from a given signature. We investigate the properties of these notions, leveraging results from earlier chapters. We show in particular that signature-bounded prime implicates are less well-behaved computationally than regular prime implicates.

Chapter 6. This chapter is concerned with the application of our notion of prime implicate to the area of knowledge compilation. We begin the chapter by showing why the obvious definition of prime implicate normal form in $\mathcal{K}_n$ is unsatisfactory, before proposing our own more sophisticated definition. We investigate the properties of our normal form, showing in particular that entailment between formulae in prime implicate normal form can be carried out in quadratic time using a simple structural comparison algorithm (which is quite similar to the procedure outlined

earlier in the chapter for propositional logic). We also show that uniform interpolation is tractable for formulae in our normal form. Later in the chapter, we propose an algorithm for putting formulae into prime implicate normal form, and we investigate the spatial complexity of this transformation, showing there to be an at most double-exponential blowup in formula size. We conclude the chapter with a comparison of prime implicate normal form to existing normal forms for $\mathcal{K}_n$ formulae.

Chapter 7. In this chapter, we summarize the main contributions of the thesis and indicate some interesting avenues of future research.

Appendix A. We provide in this appendix a brief review of computational complexity theory, in which we recall the definitions of the different complexity classes appearing in this thesis.

Relevant publications

Some of the results presented in this thesis have been previously published:

- A complete version of Chapters 3 and 4 can be found in the journal paper [Bie09]. Many of the results of Chapter 3 and some results from Chapter 4 first appeared in an earlier conference paper [Bie07b] (but with some errors).
- Some parts of Chapter 5 were presented in the workshop paper [Bie07a].
- Many of the results in Chapter 6, rephrased in terms of the description logic $\mathcal{ALC}$, were published in [Bie08b] (and were also presented in [Bie08c]).

Two related publications which were obtained during the author's doctoral studies but are not presented in this thesis are:

- [BHQ08], which introduces a prime-implicate based revision procedure
- [Bie08a], which presents complexity results for abductive reasoning in the $\mathcal{EL}$ family of description logics

2

The Modal Logic $\mathcal{K}_n$

In this chapter, we introduce the basics of the modal logic $\mathcal{K}_n$ [1]. We begin by recalling the syntax and semantics of the logic $\mathcal{K}_n$ and introducing some key notation and terminology. Next, we highlight some properties of logical consequence in $\mathcal{K}_n$ which will prove useful to us in later chapters. After that, we introduce some basic transformations and reasoning tasks in $\mathcal{K}_n$ and study their properties. Finally, at the end of the chapter, we discuss the relationship of the modal logic $\mathcal{K}_n$ with first-order logic and with description logics.

Several of the results presented in this chapter have appeared previously in the literature, but we have chosen to include proofs of these results in order to make this thesis as self-contained as possible.

2.1 Syntax

Formulae in $\mathcal{K}_n$ are built up from a set of propositional variables $\mathcal{V}$, the standard logical connectives ($\neg$, $\wedge$, and $\vee$), and the modal operators $\Box_i$ and $\Diamond_i$ (for $1 \leq i \leq n$). For convenience, we will also include the special zero-ary connectives $\top$ and $\bot$ to represent the tautology and contradiction, and we will often treat $\wedge$ and $\vee$ as multiple-arity connectives. In the special case where $n = 1$, we will write $\mathcal{K}$ instead of $\mathcal{K}_1$, and we will use $\Box$ and $\Diamond$ in place of $\Box_1$ and $\Diamond_1$.

Where convenient, we will use $\varphi \rightarrow \psi$ as an abbreviation for $\neg\varphi \vee \psi$. We adopt the shorthand $\Box_i^k\varphi$ (resp. $\Diamond_i^k\varphi$) to refer to the formula consisting of φ preceded by k copies of $\Box_i$ (resp. $\Diamond_i$), with the convention that $\Box_i^0\varphi = \Diamond_i^0\varphi = \varphi$.

1. Refer to [BdV01], [Che80], or [BvW06] for good introductions to modal logic.

We will use $var(\varphi)$ to refer to the set of propositional variables appearing in a formula φ. A *signature* for $\mathcal{K}_n$ is defined to be a subset of $\mathcal{V} \cup \{1, 2, ..., n\}$. For example, if $\mathcal{V} = \{a, b, c\}$ and $n = 3$, then both $\{1, 3, a\}$ and $\{a, b, c\}$ would be valid signatures. The *signature of a formula* φ, written $sig(\varphi)$, is defined to be the union of $var(\varphi)$ and the set of numbers j such that $\Box_j$ or $\Diamond_j$ appears in φ. For example, we have $sig(\Box_1\Diamond_2(a \vee c)) = \{1, 2, a, c\}$.

The *(modal) depth* of a formula φ, written $\delta(\varphi)$, is defined as the maximal number of nested modal operators appearing in φ, e.g. $\delta(\Diamond(a \wedge \Box a) \vee a) = 2$. We define the *length* (or size) of a formula φ, written $|\varphi|$, to be the number of occurrences of propositional variables, logical connectives, and modal operators in φ. So for example, we would have $|(a \wedge \neg b)| = 4$ and $|\Diamond_1(a \vee b) \wedge \Box_2 a| = 7$.

The set of *subformulae* of a formula φ, denoted $Sub(\varphi)$, is defined recursively as follows:

$Sub(\top) = \{\top\}$ $\quad$ $Sub(\bot) = \{\bot\}$

$Sub(a) = \{a\}$(for $a \in \mathcal{V}$) $\quad$ $Sub(\neg\psi) = \{\neg\psi\} \cup Sub(\psi)$

$Sub(\psi_1 \wedge \psi_2) = \{\psi_1 \wedge \psi_2\} \cup Sub(\psi_1) \cup Sub(\psi_2)$ $\quad$ $Sub(\Box_i\psi) = \{\Box_i\psi\} \cup Sub(\psi)$

$Sub(\psi_1 \vee \psi_2) = \{\psi_1 \vee \psi_2\} \cup Sub(\psi_1) \cup Sub(\psi_2)$ $\quad$ $Sub(\Diamond_i\psi) = \{\Diamond_i\psi\} \cup Sub(\psi)$

For example, the subformulae of $\neg(a \vee \Box_2(b \wedge \Diamond_1\top))$ are: $\neg(a \vee \Box_2(b \wedge \Diamond_1\top))$, $a \vee \Box_2(b \wedge \Diamond_1\top)$, a, $\Box_2(b \wedge \Diamond_1\top)$, $b \wedge \Diamond_1\top$, b, $\Diamond_1\top$, and $\top$. It is easily shown by induction that the cardinality of $Sub(\varphi)$ can never exceed $|\varphi|$.

An occurrence of a subformula σ of a formula φ is said to be *in the scope of a modal operator* q just in the case that there is a subformula $q\chi$ of φ such that χ contains the occurrence of σ. For instance, the first occurrence of the subformula a in $(\Box_1\Diamond_2 a) \vee \neg a$ is in the scope of the modal operators $\Box_1$ and $\Diamond_2$, but the second occurrence of a is outside the scope of any modal operators.

We will call a formula of the form $\Box_i\varphi$ (resp. $\Diamond_i\varphi$) a $\Box$-*formula* (resp. $\Diamond$-*formula*). We will say that a formula is *basic* if it is either a propositional literal or a $\Box$- or $\Diamond$-formula. A formula will be called *disjunctive* if it is a disjunction of basic formulae. A formula is said to be *conjunctive* if it is a conjunction of basic formulae.

We introduce some notation in order to refer to different components of disjunctive and conjunctive formulae. If φ is a disjunctive (resp. conjunctive) formula, then $Prop(\varphi)$ is defined to be the set of propositional literals which are disjuncts (resp. conjuncts) of φ. If φ is a disjunctive (resp. conjunctive) formula, then $Box_i(\varphi)$ is defined to be the set of formulae ψ such that $\Box_i\psi$ is a disjunct (resp. conjunct) of φ. Similarly, $Diam_i(\varphi)$ is defined as the set of formulae ψ such that $\Diamond_i\psi$ is a disjunct (resp. conjunct) of φ. For example, for the disjunctive formula $\varphi = \neg a \vee \neg b \vee \Box_1 c \vee \Box_1\Diamond_2\top \vee \Diamond_1\top$, we have $Prop(\varphi) = \{\neg a, \neg b\}$,

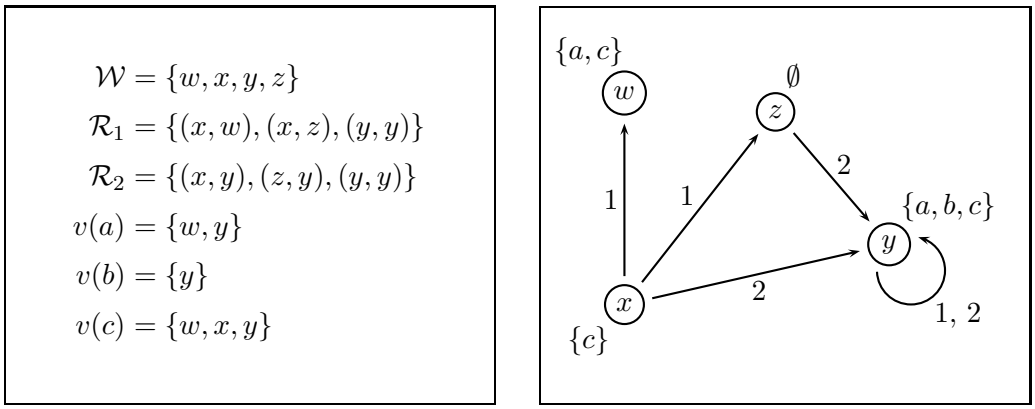

Figure 2.1: An example model and its graphical representation.

$Box_1(\varphi) = \{c, \Diamond_2\top\}$, $Box_2(\varphi) = Diam_2(\varphi) = \emptyset$, and $Diam_1(\varphi) = \{\top\}$. For the conjunctive formula $\psi = c \wedge \Box_1\bot \wedge \Box_2 a \wedge \Diamond_2(a \vee \Box_1 b) \wedge \Diamond_2 c$, we have $Prop(\psi) = \{c\}$, $Box_1(\psi) = \{\bot\}$, $Box_2(\psi) = \{a\}$, $Diam_1(\psi) = \emptyset$, and $Diam_2(\psi) = \{a \vee \Box_1 b, c\}$.

A $\mathcal{K}_n$-formula is said to be in *negation normal form* (NNF) just in the case that it does not contain $\rightarrow$ and every negation symbol appears directly in front of propositional variables. Every formula in $\mathcal{K}_n$ can be transformed in linear time into an equivalent formula in NNF of the same modal depth via a straightforward application of the standard logical equivalences. More details on the NNF transformation can be found in Section 2.4.

2.2 Semantics

A *model* (or interpretation) for $\mathcal{K}_n$ is a tuple $\mathfrak{M} = \langle \mathcal{W}, \{\mathcal{R}_i\}_{i=1}^n, v \rangle$, where $\mathcal{W}$ is a non-empty set of possible worlds, each $\mathcal{R}_i \subseteq \mathcal{W} \times \mathcal{W}$ is a binary relation over worlds, and $v : \mathcal{V} \rightarrow 2^{\mathcal{W}}$ defines for each propositional variable the set of worlds in which the variable holds. Models can be seen as labelled directed graphs, in which the vertices correspond to the elements of $\mathcal{W}$, the directed edges represent the binary relations, and the vertices are labelled by the set of propositional variables which hold in the corresponding possible world. In Figure 2.1, we give an example of a model and its corresponding graphical representation.

Satisfaction of a formula φ in a model $\mathfrak{M}$ at the world w (written $\mathfrak{M}, w \models \varphi$) is defined inductively as follows:

- $\mathfrak{M}, w \models \top$
- $\mathfrak{M}, w \not\models \bot$

- $\mathfrak{M}, w \models a$ if and only if $w \in v(a)$
- $\mathfrak{M}, w \models \neg\varphi$ if and only if $\mathfrak{M}, w \not\models \varphi$
- $\mathfrak{M}, w \models \varphi \wedge \psi$ if and only if $\mathfrak{M}, w \models \varphi$ and $\mathfrak{M}, w \models \psi$
- $\mathfrak{M}, w \models \varphi \vee \psi$ if and only if $\mathfrak{M}, w \models \varphi$ or $\mathfrak{M}, w \models \psi$
- $\mathfrak{M}, w \models \Box_i \varphi$ if and only if $\mathfrak{M}, w' \models \varphi$ for every w' such that $(w, w') \in \mathcal{R}_i$
- $\mathfrak{M}, w \models \Diamond_i \varphi$ if and only if $\mathfrak{M}, w' \models \varphi$ for some w' such that $(w, w') \in \mathcal{R}_i$

If we think of models as labelled directed graphs, then determining the satisfaction of a formula $\Box_i \varphi$ at vertex w consists in evaluating φ at all of the vertices which can be reached from w via an i-labelled edge; $\Box_i \varphi$ is satisfied at w just in the case that φ holds in each of these successor vertices. Similarly, in order to decide whether a formula $\Diamond_i \varphi$ holds at a vertex w, we consider each of the i-successors of w in the graph and check whether at least one of these vertices satisfies φ.

Example 2.2.1.
Let $\mathfrak{M}$ be the model defined in Figure 2.1. We have:

- $\mathfrak{M}, w \models a$, since $w \in v(a)$
- $\mathfrak{M}, w \models \Box_1 \bot$, since there is no world u such that $(w, u) \in \mathcal{R}_1$
- $\mathfrak{M}, w \models a \wedge \Box_1 \bot$, since both $\mathfrak{M}, w \models a$ and $\mathfrak{M}, w \models \Box_1 \bot$
- $\mathfrak{M}, z \models \neg a$, since $\mathfrak{M}, z \not\models a$
- $\mathfrak{M}, x \models \Diamond_1 \neg a$, since $(x, z) \in \mathcal{R}_1$ and $\mathfrak{M}, z \models \neg a$
- $\mathfrak{M}, x \models \Box_1 \Box_1 \bot$, since w and z are the only 1-successors of x, and both $\mathfrak{M}, w \models \Box_1 \bot$ and $\mathfrak{M}, z \models \Box_1 \bot$
- $\mathfrak{M}, x \models \Box_2 (a \wedge b \wedge c)$, since the only 2-successor of x is y and $\mathfrak{M}, y \models a \wedge b \wedge c$

A formula φ is said to be a *tautology*, written $\models \varphi$, if $\mathfrak{M}, w \models \varphi$ for every model $\mathfrak{M}$ and world w. A formula φ is *satisfiable* if there is some model $\mathfrak{M}$ and some world w such that $\mathfrak{M}, w \models \varphi$. If there is no $\mathfrak{M}$ and w for which $\mathfrak{M}, w \models \varphi$, then φ is called *unsatisfiable*, and we write $\varphi \models \bot$.

2.3 Logical Consequence

In modal logic, there are two different ways of defining logical consequence (cf. [van83] for discussion):

- a formula ψ is a *global consequence* of φ if whenever $\mathfrak{M}, w \models \varphi$ for every world w of a model $\mathfrak{M}$, then $\mathfrak{M}, w \models \psi$ for every world w of $\mathfrak{M}$
- a formula ψ is a *local consequence* of φ if $\mathfrak{M}, w \models \varphi$ implies $\mathfrak{M}, w \models \psi$ for every model $\mathfrak{M}$ and world w

In this thesis, we will be focusing on local consequence, firstly because this is the notion of consequence most often used in the modal logic literature, and secondly

because the local consequence relation is better-behaved than the global consequence relation in some important respects. In particular, the deduction theorem, familiar from classical logic, holds only with respect to the local consequence relation. In what follows, we will take $\varphi \models \psi$ to mean that ψ is a local consequence of φ, and we will say that φ (logically) *entails* ψ. Two formulae φ and ψ will be called *equivalent*, written $\varphi \equiv \psi$, if both $\varphi \models \psi$ and $\psi \models \varphi$. A formula φ is said to be *logically stronger* than ψ if $\varphi \models \psi$ and $\psi \not\models \varphi$.

In the remainder of this section, we highlight some basic properties of logical consequence in $\mathcal{K}_n$, some well-known and some less so, which will play an important role in the proofs of our results.

Theorem 2.3.1.
Let γ be a propositional formula, let ψ, χ, ψ_i, χ_i, $\psi_{i,j}$, $\chi_{i,j}$ be formulae in $\mathcal{K}_n$, and let k be an integer between 1 and n. Then

1. $\psi \models \chi \Leftrightarrow \models \neg\psi \vee \chi \Leftrightarrow \psi \wedge \neg\chi \models \bot$
2. $\Diamond_k \psi \equiv \neg\Box_k\neg\psi$
3. $\psi \models \chi \Leftrightarrow \Diamond_k \psi \models \Diamond_k \chi \Leftrightarrow \Box_k \psi \models \Box_k \chi$
4. $\Box_k(\psi_1 \wedge \psi_2 \wedge ... \wedge \psi_m) \equiv \Box_k\psi_1 \wedge \Box_k\psi_2 \wedge ... \wedge \Box_k\psi_m$
5. $\Diamond_k(\psi_1 \vee \psi_2 \vee ... \vee \psi_m) \equiv \Diamond_k\psi_1 \vee \Diamond_k\psi_2 \vee ... \vee \Diamond_k\psi_m$
6. $\gamma \wedge \bigwedge_{i=1}^{n}(\Diamond_i \psi_{i,1} \wedge ... \wedge \Diamond_i \psi_{i,l_i} \wedge \Box_i\chi_{i,1} \wedge ... \wedge \Box_i \chi_{i,m_i}) \models \bot$
   $\Leftrightarrow$ $\gamma \models \bot$ *or* $\psi_{i,j} \wedge \chi_{i,1} \wedge ... \wedge \chi_{i,m_i} \models \bot$ *for some* $1 \leq i \leq n$ *and* $1 \leq j \leq l_i$
7. $\models \gamma \vee \bigvee_{i=1}^{n}(\Diamond_i \psi_{i,1} \vee ... \vee \Diamond_i \psi_{i,l_i} \vee \Box_i\chi_{i,1} \vee ... \vee \Box_i\chi_{i,m_i})$
   $\Leftrightarrow \models \gamma$ *or* $\models \psi_{i,1} \vee ... \vee \psi_{i,l_i} \vee \chi_{i,j}$ *for some* $1 \leq i \leq n$ *and* $1 \leq j \leq m_i$
8. $\Box_k \chi \models \Box_k \chi_1 \vee ... \vee \Box_k \chi_m \Leftrightarrow \chi \models \chi_i$ *for some* $1 \leq i \leq m$
9. $\Diamond_k \psi_1 \vee ... \vee \Diamond_k \psi_l \vee \Box_k \chi_1 \vee ... \vee \Box_k \chi_m$
   $\equiv \Diamond_k \psi_1 \vee ... \vee \Diamond_k \psi_l \vee \Box_k (\chi_1 \vee \psi_1 \vee ... \vee \psi_l) \vee ... \vee \Box_k (\chi_m \vee \psi_1 \vee ... \vee \psi_l)$

Proof. The first statement is a well-known property of local consequence, but we prove it here for completeness:

$$\begin{aligned}
\psi \models \chi \quad &\Leftrightarrow \quad \mathfrak{M}, w \models \psi \text{ implies } \mathfrak{M}, w \models \chi \text{ for all } \mathfrak{M}, w \\
&\Leftrightarrow \quad \mathfrak{M}, w \not\models \psi \text{ or } \mathfrak{M}, w \models \chi \text{ for all } \mathfrak{M}, w \\
&\Leftrightarrow \quad \mathfrak{M}, w \models \neg\psi \text{ or } \mathfrak{M}, w \models \chi \text{ for all } \mathfrak{M}, w \\
&\Leftrightarrow \quad \models \neg\psi \vee \chi \\
&\Leftrightarrow \quad \mathfrak{M}, w \not\models \psi \wedge \neg\chi \text{ for all } \mathfrak{M}, w \\
&\Leftrightarrow \quad \psi \wedge \neg\chi \models \bot
\end{aligned}$$

The second statement is also standard, and can be simply proved as follows:

$$\begin{aligned}
\mathfrak{M}, w \models \Diamond_k \psi &\Leftrightarrow \text{there is some } v \text{ such that } (w,v) \in \mathcal{R}_k \text{ and } \mathfrak{M}, v \models \psi \\
&\Leftrightarrow \text{there is some } v \text{ such that } (w,v) \in \mathcal{R}_k \text{ and } \mathfrak{M}, v \not\models \neg\psi \\
&\Leftrightarrow \mathfrak{M}, w \not\models \Box_k \neg\psi \\
&\Leftrightarrow \mathfrak{M}, w \models \neg\Box_k \neg\psi
\end{aligned}$$

For the third statement, if $\psi \not\models \chi$, then there is some $\mathfrak{M}, w$ such that $\mathfrak{M}, w \models \psi \wedge \neg\chi$. Create a new model $\mathfrak{M}'$ from $\mathfrak{M}$ by adding a new world w' and placing a single k-arc from w' to w. Then $\mathfrak{M}', w' \models \Diamond_k \psi \wedge \Box_k \neg\chi$, which means that $\Diamond_k \psi \wedge \Box_k \neg\chi$ is satisfiable and hence $\Diamond_k \psi \not\models \Diamond_k \chi$ (using statements 1 and 2). For the other direction, suppose $\Diamond_k \psi \not\models \Diamond_k \chi$. Then there exists $\mathfrak{M}, w$ such that $\mathfrak{M}, w \models \Diamond_k \psi \wedge \neg\Diamond_k \chi \equiv \Diamond_k \psi \wedge \Box_k \neg\chi$. But this means that there is some w' for which $\psi \wedge \neg\chi$, hence $\psi \not\models \chi$. To complete the proof, we use the following chain of equivalences: $\Box_k \psi \models \Box_k \chi \Leftrightarrow \neg\Box_k \chi \models \neg\Box_k \psi \Leftrightarrow \Diamond_k \neg\chi \models \Diamond_k \neg\psi \Leftrightarrow \neg\chi \models \neg\psi \Leftrightarrow \psi \models \chi$.

For statement 4, we have $\mathfrak{M}, w \models \Box_k(\psi_1 \wedge \psi_2 \wedge ... \wedge \psi_m)$ if and only if $\mathfrak{M}, w' \models \psi_1 \wedge \psi_2 \wedge ... \wedge \psi_m$ for every w' with $w\mathcal{R}_k w'$ if and only if $\mathfrak{M}, w' \models \psi_i$ for every w' with $w\mathcal{R}_k w'$ and $1 \leq i \leq m$ if and only if $\mathfrak{M}, w \models \Box_k \psi_i$ for every $1 \leq i \leq m$ if and only if $\mathfrak{M}, w \models \Box_k \psi_1 \wedge \Box_k \psi_2 \wedge ... \wedge \Box_k \psi_m$.

Statement 5 is shown as follows: $\mathfrak{M}, w \models \Diamond_k(\psi_1 \vee \psi_2 \vee ... \vee \psi_m)$ if and only if $\mathfrak{M}, w' \models \psi_1 \vee \psi_2 \vee ... \vee \psi_m$ for some w' with $w\mathcal{R}_k w'$ if and only if $\mathfrak{M}, w' \models \psi_i$ for some $1 \leq i \leq m$ and w' with $w\mathcal{R}_k w'$ if and only if $\mathfrak{M}, w \models \Diamond_k \psi_i$ for some $1 \leq i \leq m$ if and only if $\mathfrak{M}, w \models \Diamond_k \psi_1 \vee \Diamond_k \psi_2 \vee ... \vee \Diamond_k \psi_m$.

For 6, suppose $\gamma \wedge \bigwedge_{i=1}^{n}(\Diamond_i \psi_{i,1} \wedge ... \wedge \Diamond_i \psi_{i,l_i} \wedge \Box_i \chi_{i,1} \wedge ... \wedge \Box_i \chi_{i,m_i}) \not\models \bot$. Then there exist $\mathfrak{M}, w$ such that $\mathfrak{M}, w \models \gamma \wedge \bigwedge_{i=1}^{n}(\Diamond_i \psi_{i,1} \wedge ... \wedge \Diamond_i \psi_{i,l_i} \wedge \Box_i \chi_{i,1} \wedge ... \wedge \Box_i \chi_{i,m_i})$. As $\mathfrak{M}, w \models \gamma$, we cannot have $\gamma \models \bot$, nor can we have $\psi_{i,j} \wedge \chi_{i,1} \wedge ... \wedge \chi_{i,m_i} \models \bot$ since for each i and each $1 \leq j \leq l_i$ there is some w' accessible from w via an i-arc such that $\mathfrak{M}, w' \models \psi_{i,j} \wedge \chi_{i,1} \wedge ... \wedge \chi_{i,m_i}$. Now for the other direction suppose that γ and all of the formulae $\psi_{i,j} \wedge \chi_{i,1} \wedge ... \wedge \chi_{i,m_i}$ (for $1 \leq i \leq n$ and $1 \leq j \leq l_i$) are satisfiable. Then there is some propositional model w of γ, and for each pair i, j, we can find $\mathfrak{M}_{i,j}, w_{i,j}$ such that $\mathfrak{M}_{i,j}, w_{i,j} \models \psi_{i,j} \wedge \chi_{i,1} \wedge ... \wedge \chi_{i,m_i}$. Now we construct a new Kripke structure which contains the models $\mathfrak{M}_{i,j}$ and the world w and in which there are i-arcs going from w to each of the $w_{i,j}$. It can be easily verified that this new model $\mathfrak{M}_{new}$ is such that $\mathfrak{M}_{new}, w \models \gamma \wedge \bigwedge_{i=1}^{n}(\Diamond_i \psi_{i,1} \wedge ... \wedge \Diamond_i \psi_{i,l_i} \wedge \Box_i \chi_{i,1} \wedge ... \wedge \Box_i \chi_{i,m_i})$, which means this formula is satisfiable.

Statement 7 follows easily from the sixth statement. We simply notice that $\gamma \vee \bigvee_{i=1}^{n}(\Diamond_i \psi_{i,1} \vee ... \vee \Diamond_i \psi_{i,l_i} \vee \Box_i \chi_{i,1} \vee ... \vee \Box_i \chi_{i,m_i})$ is a tautology just in the case that its negation $\neg\gamma \wedge \bigwedge_{i=1}^{n}(\Box_i \neg\psi_{i,1} \wedge ... \wedge \Box_i \neg\psi_{i,l_i} \wedge \Diamond_i \neg\chi_{i,1} \wedge ... \wedge \Diamond_i \neg\chi_{i,m_i})$ is unsatisfiable.

For 8, we use statements 1 and 7 to get the following chain of equivalences:

$$\begin{aligned} & \Box_k \chi \models \Box_k \chi_1 \vee ... \vee \Box_k \chi_m \\ \Leftrightarrow \quad & \models \Diamond_k \neg\chi \vee \Box_k \chi_1 \vee ... \vee \Box_k \chi_m \\ \Leftrightarrow \quad & \models \neg\chi \vee \chi_i \text{ for some } 1 \leq i \leq m \\ \Leftrightarrow \quad & \chi \models \chi_i \text{ for some } 1 \leq i \leq m \end{aligned}$$

The first implication of the equivalence in 9 is immediate since $\Diamond_k \psi_1 \vee ... \vee \Diamond_k \psi_l \models \Diamond_k \psi_1 \vee ... \vee \Diamond_k \psi_l$ and $\Box_k \chi_i \models \Box_k (\chi_i \vee \psi_1 \vee ... \vee \psi_l)$ for all $1 \leq i \leq m$. For the other direction, we remark that by using statements 1, 2, and 6, we get the following equivalences:

$$\begin{aligned} & \Box_k (\chi_i \vee \psi_1 \vee ... \vee \psi_l) \models \Box_k \chi_i \vee \Diamond_k \psi_1 \vee ... \vee \Diamond_k \psi_l \\ \Leftrightarrow \quad & \Box_k (\chi_i \vee \psi_1 \vee ... \vee \psi_l) \wedge \neg(\Box_k \chi_i \vee \Diamond_k \psi_1 \vee ... \vee \Diamond_k \psi_l) \models \bot \\ \Leftrightarrow \quad & \Box_k (\chi_i \vee \psi_1 \vee ... \vee \psi_l) \wedge \Diamond_k \neg\chi_i \wedge \Box_k \neg\psi_1 \wedge ... \wedge \Box_k \neg\psi_l \models \bot \\ \Leftrightarrow \quad & (\chi_i \vee \psi_1 \vee ... \vee \psi_l) \wedge \neg\chi_i \wedge \neg\psi_1 \wedge ... \wedge \neg\psi_l \models \bot \end{aligned}$$

As $(\chi_i \vee \psi_1 \vee ... \vee \psi_l) \wedge \neg\chi_i \wedge \neg\psi_1 \wedge ... \wedge \neg\psi_l$ is clearly unsatisfiable, it follows that $\Box_k (\chi_i \vee \psi_1 \vee ... \vee \psi_l) \models \Box_k \chi_i \vee \Diamond_k \psi_1 \vee ... \vee \Diamond_k \psi_l$ for every i and hence that $\Diamond_k \psi_1 \vee ... \vee \Diamond_k \psi_l \vee \Box_k (\chi_1 \vee \psi_1 \vee ... \vee \psi_l) \vee ... \vee \Box_k (\chi_m \vee \psi_1 \vee ... \vee \psi_l) \models \Diamond_k \psi_1 \vee ... \vee \Diamond\psi_l \vee \Box_k \chi_1 \vee ... \vee \Box_k \chi_m$. □

Statement 1 of Theorem 2.3.1 shows us how the three reasoning tasks of deduction, unsatisfiability, and tautology-testing can be rephrased in terms of one another. The second statement shows how the $\Diamond$ and $\Box$ modal operators can be rephrased in terms of one another. Statement 3 tells us how entailment between two $\Box$- or $\Diamond$-formulae can be reduced to entailment between those formulae with the first modality removed. Statement 4 states the distributivity of conjunction over universal modalities, whereas statement 5 gives the distributivity of disjunction over existential modalities. Statements 6 and 7 define the conditions under which a conjunctive (resp. disjunctive) formula is unsatisfiable (resp. a tautology). Statement 8 gives us the conditions under which a $\Box$-formula implies a disjunction of $\Box$-formulae. Statement 9 demonstrates the interaction between $\Box$- and $\Diamond$-formulae in a disjunctive formula.

The next two theorems concern entailment between disjunctive formulae. Theorem 2.3.2 tells us what kinds of disjunctive formulae can entail a propositional clause, a disjunction of $\Diamond$-formulae, or a disjunction of $\Box$-formulae, while Theorem 2.3.3 outlines the conditions under which two disjunctive formulae can be related to each other by the entailment relation.

Theorem 2.3.2.
Let λ be a disjunctive formula in $\mathcal{K}_n$. Then each of the following statements holds:

1. *If* $\lambda \models \gamma$ *for some non-tautological propositional clause* γ*, then every disjunct of* λ *is either a propositional literal or a formula* $\Diamond_i \psi$ *where* $\psi \models \bot$
2. *If* $\lambda \models \Diamond_i \psi_1 \vee ... \vee \Diamond_i \psi_l$*, then every disjunct of* λ *must be of the form* $\Diamond_i \psi$
3. *If* $\lambda \models \Box_i \chi_1 \vee ... \vee \Box_i \chi_m$ *and* $\not\models \Box_i \chi_1 \vee ... \vee \Box_i \chi_m$*, then every disjunct of* λ *is either a formula of the form* $\Box_i \chi$ *or a formula* $\Diamond_j \psi$ *where* $\psi \models \bot$

Proof. For (1), let γ be a non-tautologous propositional clause such that $\lambda \models \gamma$, and suppose for a contradiction that λ contains a disjunct $\Box_i \chi$ or a disjunct $\Diamond_i \psi$ where $\psi \not\models \bot$. In the first case, we have $\Box_i \chi \models \gamma$, and hence $\models \Diamond_i \neg \chi \vee \gamma$. It follows from Theorem 2.3.1 that $\models \gamma$, contradicting our assumption that γ is not a tautology. In the second case, we have $\Diamond_i \psi \models \gamma$, and hence $\models \Box_i \neg \psi \vee \gamma$. By Theorem 2.3.1, either $\models \neg \psi$ or $\models \gamma$. In both cases, we reach a contradiction since we have assumed that $\psi \not\models \bot$ and $\not\models \gamma$. It follows then that λ cannot have any $\Box$-formulae or any satisfiable $\Diamond$-formulae as disjuncts.

The proofs of (2) and (3) proceed similarly. □

Theorem 2.3.3.
Let $\lambda =$

$$\gamma \vee \bigvee_{i=1}^{n} (\Diamond_i \psi_{i,1} \vee ... \vee \Diamond_i \psi_{i,l_i} \vee \Box_i \chi_{i,1} \vee ... \vee \Box_i \chi_{i,m_i})$$

and $\lambda' =$

$$\gamma' \vee \bigvee_{i=1}^{n} (\Diamond_i \psi'_{i,1} \vee ... \vee \Diamond_i \psi'_{i,p_i} \vee \Box_i \chi'_{i,1} \vee ... \vee \Box_i \chi'_{i,q_i})$$

be formulae in $\mathcal{K}_n$*. If* γ *and* γ' *are both propositional and* $\not\models \lambda'$*, then* $\lambda \models \lambda'$ *if and only if the following three conditions hold:*

1. $\gamma \models \gamma'$
2. *for every* $1 \leq i \leq n$*:* $\psi_{i,1} \vee ... \vee \psi_{i,l_i} \models \psi'_{i,1} \vee ... \vee \psi'_{i,p_i}$
3. *for every* $1 \leq i \leq n$ *and every* $1 \leq j \leq m_i$*: there is some* $1 \leq k \leq q_i$ *such that* $\chi_{i,j} \models \psi'_{i,1} \vee ... \vee \psi'_{i,p_i} \vee \chi'_{i,k}$

Proof. Since we have $\not\models \lambda'$, we know by Theorem 2.3.1 that $\not\models \gamma'$ and that $\not\models \psi'_{i,1} \vee ... \vee \psi'_{i,p_i} \vee \chi'_{i,k}$ for all values of i and k. Using this information together with

Theorem 2.3.1, we obtain the following equivalences, for $1 \leq i \leq n$ and $1 \leq j \leq m_i$:

$$\begin{array}{rcl}
\gamma \models \lambda' & \Leftrightarrow & \models \neg\gamma \vee \gamma' \vee \\
 & & \quad \bigvee_{i=1}^{n} (\Diamond_i \psi'_{i,1} \vee ... \vee \Diamond_i \psi'_{i,p_i} \vee \Box_i \chi'_{i,1} \vee ... \vee \Box_i \chi'_{i,q_i}) \\
 & \Leftrightarrow & \models \neg\gamma \vee \gamma' \\
 & \Leftrightarrow & \gamma \models \gamma' \\
\Diamond_i \psi_{i,1} \vee ... \vee \Diamond \psi_{i,l_i} \models \lambda' & \Leftrightarrow & \Diamond_i(\psi_{i,1} \vee ... \vee \psi_{i,l_i}) \models \lambda' \\
 & \Leftrightarrow & \models \gamma' \vee \Box_i \neg(\psi_{i,1} \vee ... \vee \psi_{i,l_i}) \vee \\
 & & \quad \bigvee_{i=1}^{n} (\Diamond_i \psi'_{i,1} \vee ... \vee \Diamond_i \psi'_{i,p_i} \vee \Box_i \chi'_{i,1} \vee ... \vee \Box_i \chi'_{i,q_i}) \\
 & \Leftrightarrow & \models \neg(\psi_{i,1} \vee ... \vee \psi_{i,l_i}) \vee \psi'_{i,1} \vee ... \vee \psi'_{i,p_i} \\
 & \Leftrightarrow & \psi_{i,1} \vee ... \vee \psi_{i,l_i} \models \psi'_{i,1} \vee ... \vee \psi'_{i,p_i} \\
\Box \chi_{i,j} \models \lambda' & \Leftrightarrow & \models \gamma' \vee \Diamond \neg \chi_{i,j} \vee \\
 & & \quad \bigvee_{i=1}^{n} (\Diamond_i \psi'_{i,1} \vee ... \vee \Diamond_i \psi'_{i,p_i} \vee \Box_i \chi'_{i,1} \vee ... \vee \Box_i \chi'_{i,q_i}) \\
 & \Leftrightarrow & \models \psi'_{i,1} \vee ... \vee \psi'_{i,p_i} \vee \neg \chi_{i,j} \vee \chi'_{i,k} \text{ for some } k \\
 & \Leftrightarrow & \chi_{i,j} \models \psi'_{i,1} \vee ... \vee \psi'_{i,p_i} \vee \chi'_{i,j} \text{ for some } k
\end{array}$$

To complete the proof, we use the fact $\lambda \models \lambda'$ if and only if $\gamma \models \lambda'$, $\Diamond_i \psi_{i,1} \vee ... \vee \Diamond_i \psi_{i,l_i} \models \lambda'$ for every $1 \leq i \leq n$, and $\Box_i \chi_{i,j} \models \lambda'$ for every $1 \leq i \leq n$ and $1 \leq j \leq m_i$. □

We illustrate Theorem 2.3.3 on a small example.

Example 2.3.4.
Consider the formula $\lambda = \neg b \vee \Diamond(a \wedge \Diamond c) \vee \Diamond(d \wedge \Box a) \vee \Box(c \vee d)$. Then according to Theorem 2.3.3, we have:

- $\lambda \models \neg b \vee \neg d \vee \Diamond(a \vee d) \vee \Box c$, since $\neg b \models \neg b \vee \neg d$ and $(a \wedge \Diamond c) \vee (d \wedge \Box a) \models a \vee d$ and $c \vee d \models c \vee (a \vee d)$
- $\lambda \not\models a \vee \Diamond c$, since $\neg b \not\models a$
- $\lambda \not\models a \vee \neg b \vee \Diamond(a \wedge c)$, since $(a \wedge \Diamond c) \vee (d \wedge \Box a) \not\models a \wedge c$
- $\lambda \not\models \neg b \vee \Diamond(a \vee \Box a) \vee \Box c$, since $c \vee d \not\models c \vee (a \vee \Box a)$

2.4 Basic Transformations

Very often it will prove convenient to us to work with formulae having a certain syntactic form. In this section, we introduce three procedures for transforming formulae into equivalent formulae with a special syntactic structure: a procedure **Nnf** for putting formulae into negation normal form, a procedure **Dnf** for rewriting formulae as equivalent disjunctions of conjunctive formulae, and a procedure **Cnf** which transforms formulae into equivalent conjunctions of disjunctive formulae.

The Nnf transformation

We present here the standard transformation **Nnf** for putting formulae into negation normal form (cf. [DLN+92]).

Algorithm 2.1 Nnf

Input: a formula φ

Output: a formula in NNF equivalent to φ

Case 1: $\varphi = a$ or $\varphi = \neg a$. Return φ.
Case 2: $\varphi = \varphi_1 \star \varphi_2$, where $\star$ is $\vee$ or $\wedge$. Return $\mathbf{Nnf}(\varphi_1) \star \mathbf{Nnf}(\varphi_2)$.
Case 3: $\varphi = \triangle\psi$ where $\triangle$ is $\Box_i$ or $\Diamond_i$. Return $\triangle\mathbf{Nnf}(\psi)$.
Case 4a: $\varphi = \neg\neg\psi$. Return $\mathbf{Nnf}(\psi)$.
Case 4b: $\varphi = \neg(\varphi_1 \vee \varphi_2)$. Return $\mathbf{Nnf}(\neg\varphi_1) \wedge \mathbf{Nnf}(\neg\varphi_2)$.
Case 4c: $\varphi = \neg(\varphi_1 \wedge \varphi_2)$. Return $\mathbf{Nnf}(\neg\varphi_1) \vee \mathbf{Nnf}(\neg\varphi_2)$.
Case 4d: $\varphi = \neg\Box_i\,\psi$. Return $\Diamond_i\,\mathbf{Nnf}(\neg\psi)$.
Case 4e: $\varphi = \neg\Diamond_i\,\psi$. Return $\Box_i\,\mathbf{Nnf}(\neg\psi)$.

We illustrate the **Nnf** transformation on a simple example:

Example 2.4.1.
We apply the transformation **Nnf** to the formula $\neg\Box(a \wedge \Diamond(\neg b \vee c))$:

$$\begin{aligned}
\mathbf{Nnf}(\neg\Box(a \wedge \Diamond(\neg b \vee c))) &= \Diamond\mathbf{Nnf}(\neg(a \wedge \Diamond(\neg b \vee c))) \\
&= \Diamond(\mathbf{Nnf}(\neg a) \vee \mathbf{Nnf}(\neg(\Diamond(\neg b \vee c)))) \\
&= \Diamond(\neg a \vee \Box\mathbf{Nnf}(\neg(\neg b \vee c))) \\
&= \Diamond(\neg a \vee \Box(\mathbf{Nnf}(\neg\neg b) \wedge \mathbf{Nnf}(\neg c))) \\
&= \Diamond(\neg a \vee \Box(b \wedge \neg c))
\end{aligned}$$

We recall some basic properties of **Nnf**:

Theorem 2.4.2.
The output of $\mathbf{Nnf}(\varphi)$ *is a formula in negation normal form which is equivalent to* φ. *If* φ *is already in NNF, then* $\mathbf{Nnf}(\varphi) = \varphi$. *The formula* $\mathbf{Nnf}(\varphi)$ *has depth* $\delta(\varphi)$, *has signature contained in* $sig(\varphi)$, *and has length no greater than* $2|\varphi|$.

Proof. The first four properties can all be shown by very simple inductive proofs. For the fifth property, we note that the number of propositional variables remains unchanged during the execution of **Nnf**, as does the total number of binary connectives ($\wedge$, $\vee$) and modal operators. The number of negation symbols may increase, but there can be at most one negation symbol for each occurrence of a propositional variable. This means the total number of symbols in $\mathbf{Nnf}(\varphi)$ cannot exceed $2|\varphi|$.□

The Dnf transformation

We now consider the task of rewriting a formula as an equivalent disjunction of conjunctive formulae. We know from propositional logic that this transformation can require exponential time and space in the worst-case. However, for later results, it will prove important to have an algorithm which runs using only polynomial space (although its output may be exponential). For this reason, we choose to implement our transformation **Dnf** so that it returns the conjunctive formulae in the disjunction one-by-one.

Algorithm 2.2 Dnf

Input: a formula φ

Output: a set of conjunctive formulae, output one-by-one, whose disjunction is equivalent to φ

Do **Iter-Dnf**($\{$**Nnf**$(\varphi)\}$).

Algorithm 2.3 Iter-Dnf

Input: a set S of formulae in NNF

Output: a set of conjunctive formulae, output one-by-one, whose disjunction is equivalent to S

If $S = \{\psi \wedge \zeta\} \cup S'$

do **Iter-Dnf**$(S' \cup \{\psi\} \cup \{\zeta\})$

Else if $S = \{\psi \vee \zeta\} \cup S'$

do **Iter-Dnf**$(S' \cup \{\psi\})$, then do **Iter-Dnf**$(S' \cup \{\zeta\})$

Else

output $\bigwedge_{\sigma \in S} \sigma$

We demonstrate the transformation **Dnf** on an example:

Example 2.4.3.
We run **Dnf** on the formula $\varphi = a \wedge \neg(\Diamond_1 \neg b \wedge \Box_1 c) \wedge (\Diamond_1 \neg b \vee \neg \Diamond_2 \Diamond_1 \top)$. Here are the main steps in the execution of **Dnf**:

- First the function **Nnf** is called on φ, yielding the equivalent formula $a \wedge (\Box_1 b \vee \Diamond_1 \neg c) \wedge (\Diamond_1 \neg b \vee \Box_2 \Box_1 \bot)$
- Next we call **Iter-Dnf** on the singleton set $S_1 = \{a \wedge (\Box_1 b \vee \Diamond_1 \neg c) \wedge (\Diamond_1 \neg b \vee \Box_2 \Box_1 \bot)\}$
- As $S_1 = \{a \wedge ((\Box_1 b \vee \Diamond_1 \neg c) \wedge (\Diamond_1 \neg b \vee \Box_2 \Box_1 \bot))\}$, we run **Iter-Dnf** on the set $S_2 = \{a, (\Box_1 b \vee \Diamond_1 \neg c) \wedge (\Diamond_1 \neg b \vee \Box_2 \Box_1 \bot)\}$.

- As $S_2 = \{(\Box_1 b \vee \Diamond_1 \neg c) \wedge (\Diamond_1 \neg b \vee \Box_2 \Box_1 \bot)\} \cup \{a\}$, we call **Iter-Dnf** on the set $S_3 = \{a, \Box_1 b \vee \Diamond_1 \neg c, \Diamond_1 \neg b \vee \Box_2 \Box_1 \bot\}$.
- There are no $\wedge$-symbols in S_3 outside the scope of the modal operators, but there are some disjunctions remaining, so the else-if case is applicable. As $S_3 = \{\Box_1 b \vee \Diamond_1 \neg c\} \cup \{a, \Diamond_1 \neg b \vee \Box_2 \Box_1 \bot\}$, we will make two recursive subcalls. The first subcall will be on the set $S_4 = \{a, \Box_1 b, \Diamond_1 \neg b \vee \Box_2 \Box_1 \bot\}$:
  - The else-if case applies again, since $S_4 = \{\Diamond_1 \neg b \vee \Box_2 \Box_1 \bot\} \cup \{a, \Box_1 b\}$. We make two recursive subcalls to **Iter-Dnf**, the first of which is on input $S_5 = \{a, \Box_1 b, \Diamond_1 \neg b\}$:
    - There are no $\wedge$- or $\vee$-symbols outside the modal operators in S_5, so the else case applies, and we output the conjunction of elements in S_5, which is $a \wedge \Box_1 b \wedge \Diamond_1 \neg b$

    and the second of which is on input $S_6 = \{a, \Box_1 b, \Box_2 \Box_1 \bot\}$:
    - The else case applies, so we return the conjunction of elements in S_6, which is $a \wedge \Box_1 b \wedge \Box_2 \Box_1 \bot$

  The second subcall during the examination of S_3 will be on the set $S_7 = \{a, \Diamond_1 \neg c, \Diamond_1 \neg b \vee \Box_2 \Box_1 \bot\}$:
  - As $S_7 = \{\Diamond_1 \neg b \vee \Box_2 \Box_1 \bot\} \cup \{a, \Diamond_1 \neg c\}$, we are in the else-if case, so we make two recursive calls to **Iter-Dnf**. The first call will have input $S_8 = \{a, \Diamond_1 \neg c, \Diamond_1 \neg b\}$:
    - We are in the else case since S_8, so we return the conjunction $a \wedge \Diamond_1 \neg c \wedge \Diamond_1 \neg b$

    The second call to **Iter-Dnf** will be on input $S_9 = \{a, \Diamond_1 \neg c, \Box_2 \Box_1 \bot\}$:
    - We return the conjunction of elements in S_9, which is $a \wedge \Diamond_1 \neg c \wedge \Box_2 \Box_1 \bot$

We now highlight some properties of **Dnf** and **Iter-Dnf**. In the following proofs, we will use $f_{\wedge,\vee}(S)$ to denote the total number of occurrences of $\wedge$ and $\vee$ which are outside the scope of the modal operators in the set of formulae S.

Theorem 2.4.4.
Iter-Dnf *always terminates. If the input is a set S of formulae in NNF, then (a) every formula returned by* **Iter-Dnf***(S) is a conjunctive formula, and (b) the disjunction of the formulae returned by* **Iter-Dnf***(S) is equivalent to S.*

Proof. Termination is straightforward. We simply remark that $f_{\wedge,\vee}$ strictly decreases with each level of recursion of **Iter-Dnf**, and that the recursive calls stop whenever we reach a set S with $f_{\wedge,\vee}(S) = 0$.

The fact that the formulae output by **Iter-Dnf** are conjunctive formulae is immediate from the definition of **Iter-Dnf**. Indeed, only the final case in the if-

statement can produce output, and this case is only applicable when the input set contains no $\wedge$ and $\vee$ symbols outside the scope of the modal operators, i.e. when the input set consists only of propositional literals and $\Box$- and $\Diamond$-formulae.

We next show by induction on $f_{\wedge,\vee}(S)$ that the disjunction of the formulae output by **Iter-Dnf** on input S is equivalent to S. This is clearly the case when $f_{\wedge,\vee}(S) = 0$, since then there is a single output formula which is just the conjunction of elements in S. Next suppose this holds true for sets with $f_{\wedge,\vee}$-values of at most n, and let S be some set of formulae with $f_{\wedge,\vee}(S) = n + 1$. If S contains an element of the form $\psi \wedge \zeta$, then we call **Iter-Dnf** on $S \setminus \{\psi \wedge \zeta\} \cup \{\psi\} \cup \{\zeta\}$. As the latter set has a $f_{\wedge,\vee}$-values of n, we know that the disjunction of the output formulae is equivalent to $S \setminus \{\psi \wedge \zeta\} \cup \{\psi\} \cup \{\zeta\}$ and hence to S. The other case is when S possesses no conjunctions but does contain an element of the form $\psi \vee \zeta$. This means that we will call **Iter-Dnf** on inputs $S_\psi = S \setminus \{\psi \vee \zeta\} \cup \{\psi\}$ and $S_\zeta = S \setminus \{\psi \vee \zeta\} \cup \{\zeta\}$, yielding respectively sets of output formulae Σ_ψ and Σ_ζ. We can apply the induction hypothesis (applicable since $f_{\wedge,\vee}(S_\psi) = f_{\wedge,\vee}(S_\zeta) = n$) to find that the disjunction of elements in Σ_ψ (resp. Σ_ζ) is equivalent to S_ψ (resp. S_ζ). But that means that the disjunction of the output of **Iter-Dnf** on S will be the disjunction of the elements in $\Sigma_\psi \cup \Sigma_\zeta$, which we know to be equivalent to $\bigwedge S_\psi \vee \bigwedge S_\zeta$, and hence to S. □

Corollary 2.4.5.
Dnf *always terminates. On input* φ*, the procedure* **Dnf** *returns a set of conjunctive formulae whose disjunction is equivalent to* φ*.*

Theorem 2.4.6.
On input φ*, the procedure* **Dnf** *outputs at most* $2^{|\varphi|}$ *formulae, each of which has at most* $|\varphi|$ *conjuncts. If* φ *is in NNF and there are* l *mutually non-equivalent basic subformulae which appear outside the scope of* φ*'s modal operators, then there are at most* 2^l *non-equivalent formulae output by* **Dnf**(φ)*, each of which has at most* l *non-equivalent conjuncts. Each of the formulae output by* **Dnf**(φ) *has length at most* $2|\varphi|$ *(or at most* $|\varphi|$ *if* φ *is in NNF), has depth at most* $\delta(\varphi)$*, and has signature contained in* $sig(\varphi)$*.*

Proof. We remark that each call to **Iter-Dnf** can yield at most two recursive sub-calls. Moreover, the maximal recursion depth on input S is $f_{\wedge,\vee}(S)$, since we decrease the value of $f_{\wedge,\vee}$ by 1 with each call, and we stop the recursion when the value reaches 0. It follows that there can be no more than $2^{f_{\wedge,\vee}(S)}$ terminating sub-calls during the execution of **Iter-Dnf** on S, each of which may produce at most one formula. As on input φ, **Dnf** simply runs **Iter-Dnf** on **Nnf**(φ), and **Nnf**(φ) has exactly the same $f_{\wedge,\vee}$-value as φ (this is easily shown by induction),

it follows that there are no more than $2^{f_{\wedge,\vee}(\{\varphi\})}$ formulae output by $\mathbf{Dnf}(\varphi)$. As $f_{\wedge,\vee}(\{\varphi\})$ is bounded above by $|\varphi|$ by definition, we can have at most $2^{|\varphi|}$ formulae in the output of $\mathbf{Dnf}(\varphi)$.

We next remark that if during the execution of **Iter-Dnf** on a set S there is a sub-call made on a set S', then the cardinality of S' is at most one greater than the cardinality of S. As the maximal recursion depth of **Iter-Dnf** on input S is $f_{\wedge,\vee}(S)$, it follows that the formulae output by **Iter-Dnf** will have at most $|S| + f_{\wedge,\vee}(S)$ conjuncts. When we run $\mathbf{Dnf}(\varphi)$, we call **Iter-Dnf** on the set $\{\mathbf{Nnf}(\varphi)\}$. As $|\{\mathbf{Nnf}(\varphi)\}| = 1$ and $f_{\wedge,\vee}(\mathbf{Nnf}(\varphi)) = f_{\wedge,\vee}(\varphi)$, we find that there are at most $f_{\wedge,\vee}(\{\varphi\}) + 1$ conjuncts in each output formula. We now show that $f_{\wedge,\vee}(\{\varphi\})$ is strictly smaller than $|\varphi|$. This is clearly the case when $f_{\wedge,\vee}(\{\varphi\}) = 0$, since $|\varphi|$ must be positive. If $f_{\wedge,\vee}(\{\varphi\}) > 0$, then φ contains either a symbol $\wedge$ or $\vee$, in which case the conjuncts (resp. disjuncts) must contain some symbol other than $\wedge$ or $\vee$. Thus, $f_{\wedge,\vee}(\{\varphi\}) \leq |\varphi| - 1$, which gives us $f_{\wedge,\vee}(\{\varphi\}) + 1 \leq |\varphi|$, completing the proof of this statement.

For the next properties, we begin by proving (by induction on the value of $f_{\wedge,\vee}(S)$) that the conjuncts of the formulae output by $\mathbf{Iter\text{-}Dnf}(S)$ are all basic subformulae of formulae in S which appear outside the scope of modal operators. The base case is when $f_{\wedge,\vee}(S) = 0$, in which case we output just the conjunction of elements in S, each of which is a basic subformula of itself. Next suppose the result holds for $f_{\wedge,\vee}$-values of k or less, and let S be a set of formulae in NNF such that $f_{\wedge,\vee}(S) = k+1$. If $S = \{\psi \wedge \zeta\} \cup S'$, then we call **Iter-Dnf** on $S \setminus \{\psi \wedge \zeta\} \cup \{\psi\} \cup \{\zeta\}$. The latter set has an $f_{\wedge,\vee}$-value of k, so the induction hypothesis applies, allowing us to conclude that all of conjuncts of the formulae output are basic subformulae of elements in $S \setminus \{\psi \wedge \zeta\} \cup \{\psi\} \cup \{\zeta\}$ which appear outside the scope of modal operators. This is enough since S and $S \setminus \{\psi \wedge \zeta\} \cup \{\psi\} \cup \{\zeta\}$ have the same set of basic subformulae appearing outside the scope of modal operators. If instead, we have $S = \{\psi \vee \zeta\} \cup S'$, we will call **Iter-Dnf** on $S' \cup \{\psi\}$ and then on $S' \cup \{\zeta\}$. Both $S' \cup \{\psi\}$ and $S' \cup \{\zeta\}$ have $f_{\wedge,\vee}$-values of at most k, and their basic subformulae are all basic subformulae of S, so we conclude that the conjuncts of the formulae in the output of **Iter-Dnf** on S are all basic subformulae of S which appears outside the modal operators in S.

Since $\mathbf{Dnf}(\varphi)$ is just $\mathbf{Iter\text{-}Dnf}(\{\mathbf{Nnf}(\varphi)\})$, it follows that the conjuncts of the formulae output by $\mathbf{Dnf}(\varphi)$ are all basic subformulae of $\mathbf{Nnf}(\varphi)$. As $\mathbf{Nnf}(\varphi) = \varphi$ whenever φ is in NNF (Theorem 2.4.2), we can conclude that if φ is a formula in NNF with exactly l mutually non-equivalent basic subformulae, then there can be at most 2^l mutually non-equivalent formulae output by $\mathbf{Dnf}(\varphi)$, each of which has no more than l mutually non-equivalent conjuncts.

Let us define the length of a set of formulae to be the length of the conjunction of its elements. We remark that if during the execution of **Iter-Dnf** on a set S there is a sub-call made to **Iter-Dnf** on a set S', then the length of S' is never greater than that of S. This means that any formula output during the execution of **Iter-Dnf** on a set S can have length at most the length of S. As $\mathbf{Dnf}(\varphi)$ calls **Iter-Dnf** on $\{\mathbf{Nnf}(\varphi)\}$, which by Theorem 2.4.2 has length no greater than $2|\varphi|$ (resp. $|\varphi|$ if φ is in NNF), it follows that all of the formulae output by $\mathbf{Dnf}(\varphi)$ have length at most $2|\varphi|$ (resp. $|\varphi|$ if φ is in NNF).

It was shown above that the conjuncts of the formulae which are output by $\mathbf{Iter\text{-}Dnf}(\mathbf{Nnf}(\varphi))$ (and hence by $\mathbf{Dnf}(\varphi)$) are all subformulae of $\mathbf{Nnf}(\varphi)$. By Theorem 2.4.2, $\mathbf{Nnf}(\varphi)$ has precisely the same depth and propositional letters as φ. It follows then that the formulae in $\mathbf{Dnf}(\varphi)$ have depth at most $\delta(\varphi)$ and contain only those propositional symbols appearing in φ. □

Theorem 2.4.7.
Dnf *runs in polynomial space in the size of its input.*

Proof. Straightforward: on input S, **Iter-Dnf** either terminates directly (possibly outputting a formula of the same length as S), or it makes recursive sub-calls on sets whose lengths are no greater than that of S. This is sufficient to show the result since **Dnf** calls **Iter-Dnf** on the set $\{\mathbf{Nnf}(\varphi)\}$ which has size at most $2|\varphi|$ (by Theorem 2.4.2). □

Theorem 2.4.8.
Dnf *runs in single-exponential time in the size of its input.*

Proof. We remark that on input S the procedure **Iter-Dnf** spends a linear amount of time examining S (in order to determine which case applies), and then proceeds either to make one or two recursive calls or output a formula. Thus, the total running time of the algorithm is proportional to the number of recursive subcalls to **Iter-Dnf**. We saw in the proof of Theorem 2.4.6 that there can be no more than $2^{|\varphi|}$ recursive subcalls when **Iter-Dnf** is run on $\{\mathbf{Nnf}(\varphi)\}$. It follows that the total execution time for **Dnf** on input φ is single-exponential in the length of φ. □

The Cnf transformation

By making minor modifications to the algorithm **Dnf** in the previous subsection, we obtain a procedure **Cnf** which transforms $\mathcal{K}_n$ formulae into equivalent conjunctions of disjunctive formulae.

Algorithm 2.4 Cnf

Input: a formula φ

Output: a set of disjunctive formulae, output one-by-one, whose conjunction is equivalent to φ

Do **Iter-Cnf**($\{$**Nnf**$(\varphi)\}$).

Algorithm 2.5 Iter-Cnf

Input: a set S of formulae in NNF

Output: a set of disjunctive formulae, output one-by-one, whose conjunction is equivalent to S

If $S = \{\psi \vee \zeta\} \cup S'$

do **Iter-Cnf**$(S' \cup \{\psi\} \cup \{\zeta\})$

Else if $S = \{\psi \wedge \zeta\} \cup S'$

do **Iter-Cnf**$(S' \cup \{\psi\})$, then do **Iter-Cnf**$(S' \cup \{\zeta\})$

Else

output $\bigvee_{\sigma \in S} \sigma$

The following results highlight some of the properties of the transformation **Cnf**. The proofs of these results are omitted as they are very similar to the corresponding proofs for **Dnf**.

Theorem 2.4.9.
Cnf *always terminates. On input* φ*, the procedure* **Cnf** *returns a set of disjunctive formulae whose conjunction is equivalent to* φ*.*

Theorem 2.4.10.
Cnf *run in single-exponential time in the size of its input.*

Theorem 2.4.11.
On input φ*, the procedure* **Cnf** *outputs at most* $2^{|\varphi|}$ *formulae, each of which has at most* $|\varphi|$ *disjuncts. If* φ *is in NNF and there are* l *mutually non-equivalent basic subformulae which appear outside the scope of* φ*'s modal operators, then there are at most* 2^l *non-equivalent formulae output by* **Cnf**(φ)*, each of which has at most* l *non-equivalent disjuncts. Each of the formulae output by* **Cnf**(φ) *has length at most* $2|\varphi|$ *(or at most* $|\varphi|$ *if* φ *is in NNF), has depth at most* $\delta(\varphi)$*, and has signature contained in* $sig(\varphi)$*.*

2.5 Basic Reasoning Tasks

In this section, we study the standard reasoning problems for $\mathcal{K}_n$, which are:

Satisfiability: Is φ satisfiable?

Unsatisfiability: Is φ unsatisfiable?

Entailment: Does φ entail ψ?

The complexity of these tasks was investigated in [Lad77], where it was shown that all three tasks were PSPACE-complete. Ladner's PSPACE-hardness result was proven by means of a reduction from the validity problem for quantified boolean formulae, which is the canonical PSPACE-complete problem.

Theorem 2.5.1 ([Lad77]).
Satisfiability in $\mathcal{K}$ is PSPACE*-hard.*

Proof Sketch. We recall that a quantified boolean formula (QBF) is an expression of the form $Q_1p_1...Q_mp_m\theta$ where each Q_i is either $\exists$ or $\forall$, the p_i are distinct propositional variables, and θ is a propositional formula over variables $\{p_1, ..., p_m\}$. Validity of a QBF $\beta = Q_1p_1...Q_mp_m\theta$ is defined recursively as follows: if $Q_1 = \forall$ (resp. $Q_1 = \exists$), β is valid if and only if both (resp. either) $Q_2p_2...Q_mp_m(\theta_{p_1\leftarrow\top})$ and (resp. or) $Q_2p_2...Q_mp_m(\theta_{p_1\leftarrow\bot})$ are valid (the base case, when β is propositional, is treated as in propositional logic) [2]. The problem of deciding whether a QBF is valid was shown PSPACE-complete in [SM73].

Figure 2.2 presents an encoding of a QBF $\beta = Q_1p_1...Q_mp_m\theta$ in a $\mathcal{K}_n$-formula $f(\beta)$ that is used in [BdV01] to demonstrate the PSPACE-hardness of satisfiability in $\mathcal{K}_n$. In addition to the propositional variables p_1, ..., p_m, the formula $f(\beta)$ contains variables q_0, ..., q_m. Informally speaking, these variables are used to keep track of the number of quantifiers treated so far. We begin in q_0 (part (i) of $f(\beta)$), and we pass from q_i to q_{i+1} with each modal operator (parts (ii) and (iiia)). When the quantifier associated with the current state is universal, there must be two successor states, corresponding to the two ways of instantiating the variable p_i (part (iiib)); the choices of variable values are preserved as we pass through the different levels of quantification (part (iv)). Finally, for $f(\beta)$ to be hold, the propositional formula θ must be satisfied in all terminal states (part (v)). Thus, we find that the formula $f(\beta)$ is satisfiable just in the case that β is a QBF-validity (refer to [BdV01] for the full proof). As the formula $f(\beta)$ can be generated in polynomial time from β, and the QBF-validity problem is known to be PSPACE-hard, it follows that satisfiability of formulae in $\mathcal{K}_n$ is PSPACE-hard as well.

2. Here $\theta_{p\leftarrow\top}$ (resp. $\theta_{p\leftarrow\bot}$) denotes the formula obtained from θ by replacing all occurrences of the propositional variable p by $\top$ (resp. $\bot$).

(i) q_0
(ii) $\bigwedge_{i=0}^{m}((q_i \rightarrow \wedge_{j\neq i}\neg q_j) \wedge \Box(q_i \rightarrow \wedge_{j\neq i}\neg q_j) \wedge ... \wedge \Box^m(q_i \rightarrow \wedge_{j\neq i}\neg q_j))$
(iiia) $\bigwedge_{i=0}^{m}((q_i \rightarrow \Diamond q_{i+1}) \wedge \Box(q_i \rightarrow \Diamond q_{i+1}) \wedge ... \wedge \Box^m(q_i \rightarrow \Diamond q_{i+1}))$
(iiib) $\bigwedge_{\{i|Q_i=\forall\}} \Box^i(q_i \rightarrow (\Diamond(q_{i+1} \wedge p_{i+1}) \wedge \Diamond(q_{i+1} \wedge \neg p_{i+1})))$
(iv) $\bigwedge_{i=1}^{m-1}(\bigwedge_{j=i}^{m-1} \Box^j((p_i \rightarrow \Box p_i) \wedge (\neg p_i \rightarrow \Box\neg p_i)))$
(v) $\Box^m(q_m \rightarrow \theta)$

Figure 2.2: The formula $f(\beta)$ is the conjunction of the above formulae.

It follows from Theorem 2.5.1 that the dual problem of unsatisfiability is PSPACE-hard as well. Moreover, as satisfiability tests correspond to a special type of entailment query, the PSPACE-hardness result can also be transferred to the entailment task.

Corollary 2.5.2.
Entailment in $\mathcal{K}$ is PSPACE*-hard.*

Ladner's proof of membership of satisfiability in PSPACE was constructive: he exhibited a *tableaux-style* polynomial-space algorithm for deciding satisfiability of $\mathcal{K}_n$-formulae (cf. [HHSS06] for more discussion of different types of satisfiability algorithms for $\mathcal{K}_n$). The basic idea behind Ladner's algorithm (and tableaux-style algorithms in general) is to try to construct a model of the formula; if we succeed in constructing a model, we have proven the formula satisfiable, and if we fail to find a model (and can show that we tried all possibilities), the formula is unsatisfiable.

As satisfiability-testing appears as a component in practically all of the algorithms in this thesis, we present in some detail an algorithm for deciding satisfiability of $\mathcal{K}_n$ formulae. The algorithm, which we call **Sat**, examines each of the formulae in $\mathbf{Dnf}(\varphi)$ one-by-one. As the disjunction of the formulae in $\mathbf{Dnf}(\varphi)$ is equivalent to φ, we know that φ has a model just in the case that at least one of the formulae in $\mathbf{Dnf}(\varphi)$ has a model. Thus, we have reduced the problem of deciding satisfiability for arbitrary formula in $\mathcal{K}_n$ to the more restricted problem of deciding satisfiability for conjunctive formulae. We then exploit statement 5 of Theorem 2.3.1 which tells us that a conjunctive formula T has a model just in the case that its propositional part has a model (i.e. no complementary propositional literals) and for each conjunct $\Diamond_i\psi$ of T, the formula $\psi \wedge \bigwedge_{\chi\in Box_i(T)} \chi$ possesses a model. To check whether the latter holds, we make a recursive call to **Sat**. Termination of **Sat** follows from the fact that at each level of recursion the depth of the input formula decreases, and the recursion stops when the input formula is a

propositional formula.

Algorithm 2.6 Sat

Input: a formula φ in $\mathcal{K}_n$

Output: **yes** if φ is satisfiable, and **no** otherwise

(1) Run **Dnf**(φ), and for each output formula T, do the following:
Check whether the following conditions are verified by T:
(a) T has no conjunct $\bot$
(b) $Prop(T)$ contains no complementary literals
(c) For each conjunct $\Diamond_i\psi$ of T, **Sat**$(\psi \wedge \bigwedge_{\zeta \in Box_i(T)} \zeta)$=**yes**
Return **yes** if all three conditions hold.

(2) Return **no**.

We illustrate the functioning of the algorithm **Sat** on two small examples:

Example 2.5.3.
We use **Sat** to determine whether the formula $\varphi = a \wedge \neg(\Diamond_1\neg b \wedge \Box_1 c) \wedge (\Diamond_1\neg b \vee \neg\Diamond_2\Diamond_1\top)$ is satisfiable. In Step 1, **Sat** calls **Dnf** on input φ. We know from Example 2.4.3 that the first formula returned by **Dnf** will be $T_1 = a \wedge \Box_1 b \wedge \Diamond_1\neg b$. We examine T_1 in order to determine whether it satisfies the three conditions of Step 1. The first two conditions are verified since T_1 has no conjunct $\bot$ and no complementary propositional literal conjuncts. To check condition (c), we must call **Sat** on input $b \wedge \neg b$ because of the conjunct $\Diamond_1\neg b$. **Sat** will return **no** on this input, as there is a single formula $b \wedge \neg b$ returned by **Dnf** on input $b \wedge \neg b$, and it falsifies condition (b). This means that we will not return **yes** when examining T_1. The next formula output by **Dnf** will be $T_2 = a \wedge \Box_1 b \wedge \Box_2\Box_1\bot$. This formula satisfies all three conditions since it contains no conjunct $\bot$, no complementary literal conjuncts, and no $\Diamond$-formulae as conjuncts. This means that **Sat** will return **yes** in Step 1.

Example 2.5.4.
We use the algorithm **Sat** to test whether $\varphi = \Box_1(a \wedge b \wedge \Diamond_1\top) \wedge \Diamond_1(\neg a \vee \neg b \vee \Box_1\Diamond_2(b \wedge \bot))$ is satisfiable. In Step 1, the transformation **Dnf** is called on φ. There is a single formula in the output of **Dnf**, which is φ itself. The first two conditions are satisfied by φ since it does not contain any conjunct of the form $\bot$ nor any propositional conjuncts. In order to determine whether φ satisfies condition (c), we call **Sat** on the formula $\psi = (a \wedge b \wedge \Diamond_1\top) \wedge (\neg a \vee \neg b \vee \Box_1\Diamond_2(b \wedge \bot))$. It can be verified that the first formula in **Dnf**(ψ) is $a \wedge b \wedge \Diamond_1\top \wedge \neg a$, which falsifies condition (b). The next formula returned by **Dnf** is $a \wedge b \wedge \Diamond_1\top \wedge \neg b$, which also violates condition (b). The next and final formula in the output of **Dnf** is

$a \wedge b \wedge \Diamond_1 \top \wedge \Box_1 \Diamond_2 (b \wedge \bot)$. This formula satisfies (a) and (b) but not (c) since $\mathbf{Sat}(b \wedge \bot)$=**no** and hence $\mathbf{Sat}(\top \wedge \Diamond_2 (b \wedge \bot))$=**no**. It follows that $\mathbf{Sat}(\psi)$=**no**, which means $\mathbf{Sat}(\varphi)$=**no** as well.

Theorem 2.5.5.
The algorithm **Sat** *terminates and outputs* ***yes*** *if and only if the input formula is satisfiable.*

Proof. The proof is by induction on the depth of the input formula. We begin with the case where the input formula has depth 0. In this case, we know by Theorems 2.4.4 and 2.4.6 that the set of formulae output by $\mathbf{Dnf}(\varphi)$ is a set of propositional terms whose disjunction is equivalent to φ. If φ is satisfiable, then there must be some element T in the output of $\mathbf{Dnf}(\varphi)$ which is satisfiable. This means that when we examine T, we will find no conjunct $\bot$ nor any pair of complementary literals in $Prop(T)$, and so will return **yes**. If instead φ is unsatisfiable, then every formula in the output of $\mathbf{Dnf}(\varphi)$ must be unsatisfiable. This means that every such formula must either have a conjunct $\bot$ or contain a pair of complementary literals, so we will not return **yes** during Step 1, which means we will continue on to Step 2, where we return **no**.

Next suppose the **Sat** gives the desired result whenever the input formula has depth at most k, and consider some formula φ having depth $k+1$. In Step 1 of **Sat**, we run **Dnf** on input φ. By Theorems 2.4.4, the set of formulae output **Dnf** consists of a set of conjunctive formula whose disjunction is equivalent to φ. If φ is satisfiable, then there must be some satisfiable T which is output at some stage by **Dnf**. Since T is a satisfiable conjunctive formula, we know that it cannot contain a conjunct $\bot$, nor a pair of complementary propositional literal conjuncts, nor a conjunct $\Diamond_i \psi$ such that $\psi \wedge \bigwedge_{\zeta \in Box_i(T)} \zeta$ is unsatisfiable (by Theorem 2.3.1). But we know from Theorem 2.4.6 that T is of depth at most $k+1$, which means that if $\Diamond_i \psi$ is a conjunct of T, then $\psi \wedge \bigwedge_{\zeta \in Box_i(T)} \zeta$ must be of depth at most k. It follows that we can apply the induction hypothesis to $\psi \wedge \bigwedge_{\zeta \in Box_i(T)} \zeta$ to find that $\mathbf{Sat}(\psi \wedge \bigwedge_{\zeta \in Box_i(T)} \zeta)$=**yes**. This means that T satisfies all three conditions, and so **Sat** will return **yes** in Step 1. If instead φ is an unsatisfiable formula, then all of the formulae output by **Dnf** on input φ must themselves be unsatisfiable. There are three possibilities for every such formula T: either T has a conjunct $\bot$, or it has complementary propositional literal conjuncts, or there is some conjunct $\Diamond_i \psi$ of T such that $\psi \wedge \bigwedge_{\zeta \in Box_i(T)} \zeta$ is unsatisfiable (Theorem 2.3.1). In the first two cases, either condition (a) or (b) is falsified. In the third case, we can apply the induction hypothesis to $\psi \wedge \bigwedge_{\zeta \in Box_i(T)} \zeta$ to find that $\mathbf{Sat}(\psi \wedge \bigwedge_{\zeta \in Box_i(T)} \zeta)$=**no**, and so condition (c) is falsified. It follows that there is no output formula T satisfying

all three conditions, so **yes** will not be output in Step 1 of **Sat**, which means **no** will be returned in Step 2. □

Theorem 2.5.6.
The algorithm **Sat** *runs in polynomial space in the size of the input formula.*

Proof. We will show the result in the case that φ is in NNF. This is without loss of generality since the transformation to NNF is polynomial (see Theorem 2.4.2).

The proof is by induction on the depth of the input formula φ. The base case is when $\delta(\varphi) = 0$. In Step 1, **Sat** runs **Dnf** on φ. We know from Theorem 2.4.7 that **Dnf** requires only polynomial space in $|\varphi|$. Moreover, by Theorem 2.4.6, we know that every formula T output by **Dnf** has depth 0 and has length at most $|\varphi|$ (since φ is assumed to be in NNF). This means that testing conditions (a), (b), and (c) for some formula T in the output of **Dnf** takes linear space in $|\varphi|$. As only one formula is tested at any given time, it follows that **Sat** runs in polynomial space in $|\varphi|$.

Now suppose the result holds for formulae with depth at most k, and let φ be a formula with depth $k+1$. Now, in Step 1, **Sat** runs **Dnf** on φ. We know from Theorem 2.4.7 that running **Dnf** on φ requires only polynomial space in $|\varphi|$. Moreover, because φ is assumed to be in NNF, we know that every formula T output by **Dnf** has depth $k+1$ and has length at most $|\varphi|$ (Theorem 2.4.6). It follows that testing conditions (a) and (b) for a given T can be accomplished in linear space in $|\varphi|$. As for condition (c), we remark that if $\Diamond_i\psi$ is a conjunct of T, then the formula $\psi \wedge \bigwedge_{\zeta \in Box_i(T)} \zeta$ is a formula in NNF with depth at most k, so according to the induction hypothesis, **Sat** runs in polynomial space in $|\psi \wedge \bigwedge_{\zeta \in Box_i(T)} \zeta|$. But the length of $\psi \wedge \bigwedge_{\zeta \in Box_i(T)} \zeta$ is bounded above by the length of T, which we know to be bounded above by $|\varphi|$. It follows that condition (c) can also be checked in polynomial space in $|\varphi|$, which means that **Sat** runs in polynomial space in $|\varphi|$. □

Theorem 2.5.7 ([Lad77]).
Satisfiability and unsatisfiability of $\mathcal{K}_n$ formulae are both in PSPACE.

Proof. Follows directly from Theorems 2.5.5 and 2.5.6. □

We now introduce an algorithm **Entails** for testing entailment between $\mathcal{K}_n$ formulae. Our algorithm leverages statement 1 of Theorem 2.3.1, which tells us how entailment queries can be reformulated as unsatisfiability checks.

Theorem 2.5.8.
The algorithm **Entails** *is a sound and complete decision procedure for entailment and runs in polynomial space.*

Algorithm 2.7 Entails
Input: $\mathcal{K}_n$-formulae φ and ψ
Output: **yes** if $\varphi \models \psi$, and **no** otherwise
If **Sat**$(\varphi \wedge \neg\psi)$=**no**, then return **yes**. Otherwise, return **no**.

Proof. Direct consequence of Theorems 2.3.1, 2.5.5, and 2.5.6. □

Corollary 2.5.9.
Entailment in $\mathcal{K}_n$ is in PSPACE.

Remark 2.5.10.
For the global consequence relation, entailment in $\mathcal{K}_n$ is EXPTIME-complete (cf. [Don03]), and is therefore likely to be more difficult than entailment with respect to the local consequence relation.

2.6 Uniform Interpolation

In this section, we consider the problem of computing the finest approximation of a formula over a given signature. This task has been studied extensively in mathematical logic, where it is known as *uniform interpolation*, and in artificial intelligence, where it is commonly referred to as *(variable) forgetting* (cf. [LR94], [LLM03]).

Definition 2.6.1.
Let $\mathcal{L}$ be a signature. A formula ψ is said to be a *uniform interpolant of φ over $\mathcal{L}$*, or simply an $\mathcal{L}$-interpolant of φ, just in the case that $\varphi \models \psi$, $sig(\psi) \subseteq \mathcal{L}$, and for every ψ' such that $\varphi \models \psi'$ and $sig(\psi') \subseteq \mathcal{L}$ we have $\psi \models \psi'$.

Interestingly enough, the existence of a uniform interpolant of a formula is not guaranteed. Logics for which uniform interpolants always exist are said to have the *uniform interpolation property*[3]. Many logics do not enjoy this property, among them, classical first-order logic (cf. [Hen63]), the modal logic $\mathcal{S}4$ [GZ95], and the logic $\mathcal{K}_n$ if we use the global consequence relation (cf. [GLW06], [KWW08]).

Fortunately, the logic that we are interested in here, $\mathcal{K}_n$ with the local consequence relation, does have the uniform interpolation property. This was originally shown in [Ghi95] (see also [Vis96]). More recently, a variety of different

3. This is a stronger version of the well-known *Craig interpolation property* (cf. [Cra57]), which states that for every pair of formulae φ and ψ such that $\varphi \models \psi$, there exists a third formula χ such that $\varphi \models \chi \models \psi$ and $sig(\chi) \subseteq sig(\varphi) \cap sig(\psi)$.

procedures for constructing uniform interpolants of $\mathcal{K}_n$ formulae have been proposed (cf. [tCCMV06], [Bíl07], and [HM08]). The approach in [tCCMV06] runs in single-exponential space and produces uniform interpolants which are at most single-exponentially larger than the input formula. These complexity upper bounds are optimal, given that uniform interpolation may involve an exponential blowup in formula size in the worst-case:

Theorem 2.6.2.
The shortest $\mathcal{L}$-interpolant of a formula may be single-exponentially larger than the formula.

Proof. Direct corollary of the corresponding result for propositional formulae (cf. e.g. [LLM03]). □

We present here an alternative procedure (which is broadly similar to the one outlined in [tCCMV06]) for producing single-exponential-sized interpolants. Our algorithm **LangInt** exploits the distributivity of uniform interpolation over disjunction, shown in the following lemma:

Lemma 2.6.3.
Let $\mathcal{L}$ be a signature, and let the formulae ψ'_1, ..., ψ'_m be $\mathcal{L}$-interpolants respectively of the formulae ψ_1, ..., ψ_m. Then $\psi'_1 \vee ... \vee \psi'_m$ is an $\mathcal{L}$-interpolant of $\psi_1 \vee ... \vee \psi_m$.

Proof. Let $\mathcal{L}$, ψ_1, .., ψ_m, ψ'_1, ..., ψ'_m be as in the statement of the lemma, and let φ be such that $\psi_1 \vee ... \vee \psi_m \models \varphi$ and $sig(\varphi) \subseteq \mathcal{L}$. Then for each $1 \leq j \leq n$, we must have $\psi_j \models \varphi$, and hence $\psi'_j \models \varphi$ since we have assumed that ψ'_j is an $\mathcal{L}$-interpolant of ψ_j. But then we must also have $\psi'_1 \vee ... \vee \psi'_m \models \varphi$, completing the proof. □

Our algorithm also leverages the following lemma, which characterizes the uniform interpolants of conjunctive formulae:

Lemma 2.6.4.
Let T be a conjunctive formula, and $\mathcal{L}$ a signature. Let the formula T' be defined as follows:

- *If T is unsatisfiable, then*
  $T' = \bot$
- *Else, if $\{(\neg)v \in Prop(T) \mid v \in \mathcal{L}\} = \emptyset$ and for all $i \in \mathcal{L}$, both $Box_i(T) = \emptyset$ and $Diam_i(T) = \emptyset$, then*
  $T' = \top$

- *Otherwise:*

$$T' = (\bigwedge_{v \in Prop(T): v \in \mathcal{L}} v) \wedge (\bigwedge_{\neg v \in Prop(T): v \in \mathcal{L}} \neg v)$$
$$\wedge (\bigwedge_{i \in \mathcal{L}: Box_i(T) \neq \emptyset} \Box_i UI_{\mathcal{L}}(\bigwedge_{\chi \in Box_i(T)} \chi))$$
$$\wedge (\bigwedge_{i \in \mathcal{L}: Diam_i(T) \neq \emptyset, Box_i(T) \neq \emptyset} (\bigwedge_{\psi \in Diam_i(T)} \Diamond_i UI_{\mathcal{L}}(\psi \wedge \bigwedge_{\chi \in Box_i(T)} \chi)))$$
$$\wedge (\bigwedge_{i \in \mathcal{L}: Diam_i(T) \neq \emptyset, Box_i(T) = \emptyset} (\bigwedge_{\psi \in Diam_i(T)} \Diamond_i UI_{\mathcal{L}}(\psi)))$$

where $UI_{\mathcal{L}}(\varphi)$ is any $\mathcal{L}$-interpolant of φ.

Then T' is an $\mathcal{L}$-interpolant of T.

Proof. Let

$$T = \gamma_1 \wedge ... \wedge \gamma_k \wedge \bigwedge_{i=1}^{n} (\Diamond_i \psi_{i,1} \wedge ... \wedge \Diamond_i \psi_{i,l_i} \wedge \Box_i \chi_{i,1} \wedge ... \wedge \Box_i \chi_{i,m_i})$$

be a conjunctive formula, and let T' be as defined in the statement of the lemma. Consider some formula α with signature in $\mathcal{L}$ such that $T \models \alpha$. Because of Theorems 2.4.9 and 2.4.11, we can assume without loss of generality that α is a conjunction of disjunctive formulae with signatures in $\mathcal{L}$. Let

$$\lambda = \rho_1 \vee ... \vee \rho_p \vee \bigvee_{i \in \mathcal{L}} (\Diamond_i \epsilon_{i,1} \vee ... \vee \Diamond_i \epsilon_{i,q_i} \vee \Box_i \zeta_{i,1} \vee ... \vee \Box_i \zeta_{i,r_i})$$

be one of ψ's conjuncts. We need to show that $T' \models \lambda$, and therefore $T' \models \alpha$. If λ is tautologous, then $T' \models \lambda$ trivially holds. Likewise, if T is unsatisfiable, then we have $T' = \bot$, and hence $T' \models \lambda$. So let us now consider the case where T is satisfiable and λ non-tautologous. Since $T \models \lambda$, we know that

$$\gamma_1 \wedge ... \wedge \gamma_k \wedge \textstyle\bigwedge_{i=1}^{n} (\Diamond_i \psi_{i,1} \wedge ... \wedge \Diamond_i \psi_{i,l_i} \wedge \Box_i \chi_{i,1} \wedge ... \wedge \Box_i \chi_{i,m_i}) \wedge$$
$$\neg\rho_1 \wedge ... \wedge \neg\rho_p \wedge \textstyle\bigwedge_{i \in \mathcal{L}} (\Box_i \neg\epsilon_{i,1} \wedge ... \wedge \Box_i \neg\epsilon_{i,q_i} \wedge \Diamond_i \neg\zeta_{i,1} \wedge ... \wedge \Diamond_i \neg\zeta_{i,r_i})$$

is unsatisfiable. It follows then by Theorem 2.3.1 that one of the following holds:

(a) $\gamma_1 \wedge ... \wedge \gamma_k \wedge \neg\rho_1 \wedge ... \wedge \neg\rho_p \models \bot$

(b) there exists some $i \in \mathcal{L}$ and some $1 \leq u \leq l_i$ such that
$\psi_{i,u} \wedge \chi_{i,1} \wedge ... \wedge \chi_{i,m_i} \wedge \neg\epsilon_{i,1} \wedge ... \wedge \neg\epsilon_{i,q_i} \models \bot$

(c) there exists some $i \in \mathcal{L}$ and some $1 \leq u \leq r_i$ such that
$\neg\zeta_{i,u} \wedge \chi_{i,1} \wedge ... \wedge \chi_{i,m_i} \wedge \neg\epsilon_{i,1} \wedge ... \wedge \neg\epsilon_{i,q_i} \models \bot$

If (a) holds, then because we have assumed T satisfiable and λ non-tautologous, we know that there must be some u and v such that $\gamma_u = \rho_v$. As γ_u is a conjunct of T', we have $T' \models \lambda$.

If (b) holds, then we have $\psi_{i,u} \wedge \chi_{i,1} \wedge ... \wedge \chi_{i,m_i} \models \epsilon_{i,1} \vee ... \vee \epsilon_{i,q_i}$. As $sig(\epsilon_{i,1} \vee ... \vee \epsilon_{i,q_i}) \subseteq sig(\lambda) \subseteq \mathcal{L}$, we know that every $\mathcal{L}$-interpolant of $\psi_{i,u} \wedge \chi_{i,1} \wedge ... \wedge \chi_{i,m_i}$ must entail $\epsilon_{i,1} \vee ... \vee \epsilon_{i,q_i}$. Thus, $UI_{\mathcal{L}}(\psi_{i,u} \wedge \chi_{i,1} \wedge ... \wedge \chi_{i,m_i}) \models \epsilon_{i,1} \vee ... \vee \epsilon_{i,q_i}$, and hence $\Diamond_i UI_{\mathcal{L}}(\psi_{i,u} \wedge \chi_{i,1} \wedge ... \wedge \chi_{i,m_i}) \models \Diamond_i \epsilon_{i,1} \vee ... \vee \Diamond_i \epsilon_{i,q_i}$. As $i \in \mathcal{L}$, we know that $\Diamond_i UI_{\mathcal{L}}(\psi_{i,u} \wedge \chi_{i,1} \wedge ... \wedge \chi_{i,m_i})$ (or $\Diamond_i UI_{\mathcal{L}}(\psi_{i,u})$, if $\Box_i(T) = \emptyset$) is a conjunct of T'. This means $T' \models \lambda$.

Finally, consider the case where (c) holds. Then we have $\chi_{i,1} \wedge ... \wedge \chi_{i,m_i} \models \zeta_{i,u} \vee \epsilon_{i,1} \vee ... \vee \epsilon_{i,q_i}$. As $UI_{\mathcal{L}}(\chi_{i,1} \wedge ... \wedge \chi_{i,m_i})$ is an $\mathcal{L}$-interpolant of $\chi_{i,1} \wedge ... \wedge \chi_{i,m_i}$ and $sig(\zeta_{i,u} \vee \epsilon_{i,1} \vee ... \vee \epsilon_{i,q_i}) \subseteq sig(\lambda) \subseteq \mathcal{L}$, we must have $UI_{\mathcal{L}}(\chi_{i,1} \wedge ... \wedge \chi_{i,m_i}) \models \zeta_{i,u} \vee \epsilon_{i,1} \vee ... \vee \epsilon_{i,q_i}$, and thus $\Box_i UI_{\mathcal{L}}(\chi_{i,1} \wedge ... \wedge \chi_{i,m_i}) \models \Box_i \zeta_{i,u} \vee \Diamond_i \epsilon_{i,1} \vee ... \vee \Diamond_i \epsilon_{i,q_i}$. As $i \in \mathcal{L}$, we know that $\Box_i UI_{\mathcal{L}}(\chi_{i,1} \wedge ... \wedge \chi_{i,m_i})$ is a conjunct of T', which means $T' \models \lambda$.

We have thus shown that T' entails every formula which is implied by T and has signature in $\mathcal{L}$. As T' has signature in $\mathcal{L}$ by definition, it follows that T' is an $\mathcal{L}$-interpolant of T. □

We now present our algorithm **LangInt** for generating $\mathcal{L}$-interpolants. The idea behind our algorithm is very simple: we first rewrite the input formula as an disjunction of conjunctive formula, then we compute the $\mathcal{L}$-interpolants of each of the conjunctive formulae (using Lemma 2.6.4), and finally we take the disjunction of these $\mathcal{L}$-interpolants.

Example 2.6.5.
We run **LangInt** on the formula $\varphi = (a \wedge b \wedge \Box_2 c \wedge \Box_2 \neg c \wedge \Diamond_1 \Box_2 a) \vee (\neg a \wedge c \wedge \Diamond_2(b \wedge \Diamond_2 c) \wedge \Box_2 a)$ and signature $\{a, 2\}$:

- In Step 1, we set $\mathcal{T} = \mathbf{Dnf}(\varphi) = \{a \wedge b \wedge \Box_2(c \wedge \neg c) \wedge \Diamond_1 \Box_2 a, \neg a \wedge c \wedge \Diamond_2(b \wedge \Diamond_2 c) \wedge \Box_2 a\}$, and we initialize $\mathcal{S}$ to $\emptyset$.
- We examine the first element of $\mathcal{T}$ which is $T_1 = a \wedge b \wedge \Box_2 c \wedge \Box_2 \neg c \wedge \Diamond_1 \Box_2 a$. The condition of the if-statement does not apply since T_1 is satisfiable, so we enter the else-statement, where we initialize $NewConj$ to $\{a\}$. As $Box_2(T_1) = \{c, \neg c\} \neq \emptyset$, we make a recursive subcall on $T_2 = c \wedge \neg c$:
  - In Step 1, we compute $\mathbf{Dnf}(T_2)$, which is $\{c \wedge \neg c\}$. In Step 2, the condition of the if-statement is satisfied, so we set $T_2' = \bot$ and add it to $\mathcal{S}$. In Step 3, we output $\bot$.

  We thus add $\Box_2 \bot$ to $NewConj$. As $Diam_2(T_1) = \emptyset$, we set $T_1' = a \wedge \Box_2 \bot$ and add it to the set $\mathcal{S}$.

Algorithm 2.8 LangInt

Input: a $\mathcal{K}_n$-formula φ and a signature $\mathcal{L}$

Output: an $\mathcal{L}$-interpolant of φ

(1) Set $\mathcal{T} = \mathbf{Dnf}(\varphi)$, and initialize $\mathcal{S}$ to $\emptyset$.
(2) For each $T \in \mathcal{T}$:
  If $\mathbf{Sat}(T)$=**no**, then
    Set $T' = \bot$
  Else
    Initialize $NewConj$ to $\{(\neg)v \in Prop(T) \mid v \in \mathcal{L}\}$
    For each $1 \leq i \leq n$ such that $i \in \mathcal{L} \cap sig(\varphi)$:
      If $Box_i(T) \neq \emptyset$, then
        Add $\Box_i$ **LangInt**$(\bigwedge_{\psi \in Box_i(T)} \psi)$ to $NewConj$
        For each $\gamma \in Diam_i(T)$:
          Add $\Diamond_i$ **LangInt**$(\gamma \wedge \bigwedge_{\psi \in Box_i(T)} \psi)$ to $NewConj$
      Else if $Diam_i(T) \neq \emptyset$
        For each $\gamma \in Diam_i(T)$:
          Add $\Diamond_i$ **LangInt**(γ) to $NewConj$
    If $NewConj \neq \emptyset$, set $T' = \bigwedge_{\kappa \in NewConj} \kappa$, else set $T' = \top$
  Add T' to $\mathcal{S}$
(3) Return $\bigvee_{\sigma \in \mathcal{S}} \sigma$

- We now examine the second and last element of $\mathcal{T}$, which is $T_3 = \neg a \wedge c \wedge \Diamond_2(b \wedge \Diamond_2 c) \wedge \Box_2 a$. The else-statement applies, so we initialize $NewConj$ to $\{\neg a\}$. We have $Box_2(T_3) = \{a\} \neq \emptyset$, so we call **LangInt** on input a. It returns a, so $\Box_2 a$ is added to $NewConj$. Next we consider the conjunct $\Diamond_2(b \wedge \Diamond_2 c)$, and we make a recusive subcall on input $T_4 = b \wedge \Diamond_2 c \wedge a$:
  - The only term output by $\mathbf{Dnf}(T_4)$ is $b \wedge \Diamond_2 c \wedge a$ itself. In Step 2, the else case applies, so we set $NewConj = \{a\}$. As there are no $\Box_2$-conjuncts but there are some $\Diamond_2$-conjuncts, we are in the second else case, and we call **LangInt** on the formula c:
    * On input c and $\{a, 2\}$, the algorithm returns just $\top$, since no formulae are added to $NewConj$ during the examination of c.

    We add $\Diamond_2 \top$ to $NewConj$, and output $a \wedge \Diamond_2 \top$.

  We add $\Diamond_2(a \wedge \Diamond_2 \top)$ to $NewConj$. We then set $T_3' = \neg a \wedge \Diamond_2(a \wedge \Diamond_2 \top)$.
- In Step 4, we return the disjunction of elements in $\mathcal{S}$, which is $(a \wedge \Box_2 \bot) \vee (\neg a \wedge \Diamond_2(a \wedge \Diamond_2 \top))$.

We now formally prove the correctness of **LangInt**:

Theorem 2.6.6.
On input φ and $\mathcal{L}$, the algorithm **LangInt** *returns an $\mathcal{L}$-interpolant of φ.*

Proof. The proof is by induction on the depth of the input formula φ. We begin with the case where $\delta(\varphi) = 0$. In Step 1, we set $\mathcal{T} = \mathbf{Dnf}(\varphi)$. We know from Corollary 2.4.5 that $\mathcal{T}$ is a set of conjunctive formulae whose disjunction is equivalent to φ. We also know from Theorem 2.4.6 that the elements in $\mathcal{T}$ have depth 0, i.e. they are all propositional terms. In Step 2, for each term T in $\mathcal{T}$, we add to $\mathcal{S}$ the formula T'. There are three possibilities: either T' is $\bot$ if T is unsatisfiable, or T' is the conjunction of the propositional literal conjuncts of T whose variables belong to $\mathcal{L}$, or if there are no such conjuncts, then $T' = \top$. Because of Lemma 2.6.4, we know that T' is an $\mathcal{L}$-interpolant of T. This means that the elements in $\mathcal{S}$ at the end of Step 2 are precisely the $\mathcal{L}$-interpolants of the elements in $\mathcal{T}$. Because uniform interpolation distributes over disjunction (Lemma 2.6.3), it follows that $\bigvee_{\sigma\in\mathcal{S}} \sigma$ is an $\mathcal{L}$-interpolant of $\bigvee_{T\in\mathcal{T}} T$ and hence of the formula φ.

Next let us suppose that **LangInt** performs as desired when the input formula has depth at most d, and let φ be a formula with depth $d+1$. Again, in Step 1, we will use **Dnf** to generate a set $\mathcal{T}$ of conjunctive formulae whose disjunction is equivalent to φ (Corollary 2.4.5). In Step 2, for each $T \in \mathcal{T}$, we add the formula T' to $\mathcal{S}$, which we know from the induction hypothesis and Lemma 2.6.4 to be an $\mathcal{L}$-interpolant of T. This means that at the end of Step 2 the set $\mathcal{S}$ contains for each $T \in \mathcal{T}$ an element T' which is an $\mathcal{L}$-interpolant of T. As uniform interpolation distributes over disjunction (Lemma 2.6.3), the formula $\bigvee_{\sigma\in\mathcal{S}} \sigma$ must be an $\mathcal{L}$-interpolant of $\bigvee_{T\in\mathcal{T}} T$ and hence of φ. It follows that **LangInt**$(\varphi,\mathcal{L})$ is an $\mathcal{L}$-interpolant of φ. □

The next theorem concerns the worst-case running time of **LangInt**.

Theorem 2.6.7.
The algorithm **LangInt** *runs in single-exponential time.*

Proof. In this proof, we let q and r be polynomial functions such that **Dnf** (resp. **Sat**) terminates in at most $2^{q(l)}$ (resp. $2^{r(l)}$) time steps on input of length l. The existence of such functions is guaranteed for **Dnf** by Theorem 2.4.8 and for **Sat** by Theorem 2.5.6 together with the fact that PSPACE ⊆ EXPTIME (cf. Appendix A).

Throughout the proof, we will use $t_k(l)$ to denote the maximum execution time of **LangInt** when the input formula is in NNF and has depth k and length l. We first consider the case where the input formula is a propositional formula in NNF with length l. In this case, the algorithm spends worst-case single-exponential time in l in Step 1 to generate the formulae in $\mathcal{T}$ (Theorem 2.4.8). We know from

Theorem 2.4.6 that there can never be more than 2^l terms in $\mathcal{T}$, each having length at most l. In Step 2, for each term T in $\mathcal{T}$, we test the satisfiability of T using **Sat**. It can be easily verified that **Sat** will take only polynomial time in $|T|$ and hence in l, because T is a propositional term. If T is unsatisfiable, then we set $T' = \bot$ (this obviously takes constant time). Otherwise, we enter the else-loop and set T' equal to the conjunction of those conjuncts of T which concern variables in $\mathcal{L}$. This clearly takes linear time in T. Thus, in Step 2, we spend a polynomial amount of time on each T, and hence a single-exponential time overall. In Step 3, we output the at most single-exponentially large formula $\bigvee_{\sigma \in \mathcal{S}} \sigma$. It follows that when the input formula has size l, the algorithm **LangInt** terminates in single-exponential time in l. We can thus find some polynomial function p such that $t_0(l) \leq 2^{p(l)}$.

Now we will try to place an upper bound on $t_{k+1}(l)$. Consider some formula φ in NNF with depth $k+1$ and length $l = |\varphi|$. In Step 1, we call the procedure **Dnf** on input φ, which we know terminates in at most $2^{q(|\varphi|)} = 2^{q(l)}$ time steps. Now in Step 2, we examine each of the elements in $\mathcal{T}$ in turn. Because φ is assumed to be in NNF, we know from Theorem 2.4.6 that there can be at most 2^l elements in $\mathcal{T}$, each having length at most l. In Step 2, we examine each of the terms T in $\mathcal{T}$ in turn. We start by calling **Sat** on T, which we know terminates in at most $2^{r(|T|)} \leq 2^{r(l)}$ steps. If T is unsatisfiable, we simply set $T' = \bot$. If instead we are in the else-case, then we begin by initializing $NewConj$, which takes only linear time in $|T| \leq l$. Determining the set of indices i such that $i \in \mathcal{L} \cap sig(\varphi)$ also takes linear time in $|T| \leq l$, as does determining for a given i, the sets $Box_i(T)$ and $Diam_i(T)$. For i such that $Box_i(T) \neq \emptyset$, we add the formula $\Box_i$ **LangInt**$(\bigwedge_{\psi \in Box_i(T)} \psi)$ to $NewConj$. Computing **LangInt**$(\bigwedge_{\psi \in Box_i(T)} \psi)$ takes time at most $t_k(l)$, since $\bigwedge_{\psi \in Box_i(T)} \psi$ is a formula in NNF with depth at most k and length at most $|T| \leq l$. We must also add for each $\gamma \in Diam_i(T)$ the formula $\Diamond_i$ **LangInt**$(\gamma \wedge \bigwedge_{\psi \in Box_i(T)} \psi)$ to $NewConj$. Since $\gamma \wedge \bigwedge_{\psi \in Box_i(T)} \psi$ is a formula in NNF with depth at most k and length at most $|T| \leq l$, we know that computing **LangInt**$(\gamma \wedge \bigwedge_{\psi \in Box_i(T)} \psi)$ takes at most $t_k(l)$ time steps. Similarly, if $Box_i(T) = \emptyset$, then the computation of **LangInt**(γ) takes at most $t_k(l)$ time steps. We remark that the number of formulae added to $NewConj$ is bounded by the number of conjuncts in T and hence by l. This means that we will never call **LangInt** more than l times, and each call requires at most $t_k(l)$ time steps. Thus, the computation for a given T of T' takes $O(2^{r(l)} + l * t_k(l))$, which means that the total execution time of Step 2 is on the order of $2^l(2^{r(l)} + l * t_k(l))$. In the final step of **LangInt**, we simply return the conjunction of elements in $\mathcal{S}$. Clearly this cannot require any more time than producing $\mathcal{S}$ in Step 2. Thus, we find that:

$$t_{k+1}(l) \in O(2^{q(l)} + 2(2^l(2^{r(l)} + l * t_k(l))))$$

It follows that for $k \geq 1$, we have

$$t_k(l) \in O((2^{l+1} * l)^k 2^{p(l)} + \Sigma_{j=0}^{k-1}(2^{l+1} * l)^j * (2^{q(l)} + 2^{l+1} * 2^{r(l)}))$$

As we always have $k \leq |\varphi|$ and $l = |\varphi|$, it follows that the running time of **LangInt** on a formula φ is in

$$O((2^{|\varphi|+1} * |\varphi|)^{|\varphi|}\, 2^{p(|\varphi|)} + \Sigma_{j=0}^{|\varphi|-1}(2^{|\varphi|+1} * |\varphi|)^j * (2^{q(|\varphi|)} + 2^{|\varphi|+1} * 2^{r(|\varphi|)}))$$

The latter expression is clearly single-exponential in $|\varphi|$ since both p, q, and r are all polynomial functions. We have thus shown that for formulae in NNF the algorithm **LangInt** runs in single-exponential time. This result can then be transferred to arbitrary formulae as the NNF transformation runs in polynomial (linear) time (Theorem 2.4.2). □

Corollary 2.6.8.
The formula output by **LangInt** *is at most single-exponentially larger than the input formula.*

Proof. Direct consequence of Theorem 2.6.7. □

The following lemma shows that **LangInt** maps disjunctive formulae to disjunctive formulae. We will make use of this property in some of the proofs of results in Chapter 5.

Lemma 2.6.9.
If the formula which is input to **LangInt** *is a disjunctive formula, then the formula which is output by* **LangInt** *is also a disjunctive formula.*

Proof. Consider some disjunctive formula $\lambda = \beta_1 \vee ... \vee \beta_m$. Since each β_j is a basic formula, the set $\mathcal{T}$ computed in Step 1 of **LangInt** is equal to $\{\beta_1, ..., \beta_m\}$. Now we remark that in Step 2 of **LangInt**, the number of conjuncts in T' is never greater than the number of conjuncts in T. As each formula β_j is its own unique conjunct, we know that all of the formulae added to $\mathcal{S}$ in Step 2 are basic formulae, which means that the formula $\bigvee_{\sigma \in \mathcal{S}} \sigma$ output by **LangInt** is a disjunctive formula. □

Finally, we close this subsection with the following lemma which shows uniform interpolation distributes over the modal operators. This property will be needed in Chapter 6.

Lemma 2.6.10.
If ψ' is an $\mathcal{L}$-interpolant of ψ, and $i \in \mathcal{L}$, then the $\mathcal{L}$-interpolant of $\Box_i\psi$ (resp. $\Diamond_i\psi$) is $\Box_i\psi'$ (resp. $\Diamond_i\psi'$).

$$ST(\top, x) = \top \qquad ST(\bot, x) = \bot$$
$$ST(v_j, x) = P_j(x) \qquad ST(\neg\varphi, x) = \neg ST(\varphi, x)$$
$$ST(\varphi \wedge \psi, x) = ST(\varphi, x) \wedge ST(\psi, x) \qquad ST(\varphi \vee \psi, x) = ST(\varphi, x) \vee ST(\psi, x)$$
$$ST(\Box_i\varphi, x) = \forall y\,(R_i(x, y) \rightarrow ST(\varphi, y)) \qquad ST(\Diamond_i\varphi, x) = \exists y\,(R_i(x, y) \wedge ST(\varphi, y))$$

Figure 2.3: Embedding of the modal logic $\mathcal{K}_n$ in first-order logic. Note that the variable y used in the translation of $\Box$- and $\Diamond$- formulae must be new (i.e. not already used in the translation).

Proof. We only give the proof for $\Box$-formulae, as the proof for $\Diamond$-formulae is very similar. Consider some formula $\Box_i\psi$ and some signature $\mathcal{L}$ such that $i \in \mathcal{L}$. Let ψ' be an $\mathcal{L}$-interpolant of ψ, and let φ be such that $\Box_i\psi \models \varphi$ and $sig(\varphi) \subseteq \mathcal{L}$. Because of Theorems 2.4.9 and 2.4.11, we can assume without loss of generality that φ is of the form $\lambda_1 \wedge ... \wedge \lambda_m$ for some disjunctive formulae λ_1, ..., λ_m. We now show that $\Box_i\psi' \models \lambda_j$ for every conjunct λ_j of φ. Consider then some such λ_j. As $\Box_i\psi \models \lambda_j$, it follows from Theorem 2.3.3 that λ_j is either tautologous or of the form $\Box_i\chi_1 \vee ...\Box_i\chi_k \vee \Diamond_i\zeta_1 \vee ... \vee \Diamond_i\zeta_l$. In the first case, we clearly have $\Box_i\psi' \models \lambda_j$. In the latter case, we know from Theorem 2.3.3 that $\psi \models \chi_s \vee \zeta_1 \vee ... \vee \zeta_l$ for every $1 \leq s \leq k$. As $sig(\lambda_j) \subseteq \mathcal{L}$ and ψ' is an $\mathcal{L}$-interpolant of ψ, we get that $\psi' \models \chi_s \vee \zeta_1 \vee ... \vee \zeta_l$ for every $1 \leq s \leq k$. But it must then be the case that $\Box_i\psi' \models \Box_i\chi_1 \vee ...\Box_i\chi_k \vee \Diamond_i\zeta_1 \vee ... \vee \Diamond_i\zeta_l$ (Theorem 2.3.3). We have thus shown that $\Box_i\psi \models \lambda_j$ for each $1 \leq j \leq n$, and hence that $\Box_i\psi \models \varphi$. This means that $\Box_i\psi'$ is an $\mathcal{L}$-interpolant of $\Box_i\psi$.

2.7 Relation to First-Order Logic

As we mentioned in Chapter 1, modal logics generally correspond to fragments of classical first-order logic. In this section, we examine more closely the relationship holding between the modal logic $\mathcal{K}_n$ and first-order logic.

In Figure 2.3, we present the *standard translation* of $\mathcal{K}_n$-formulae into first-order logic formulae (cf. [van83], [Bv06], [HHSS06]). We remark that each propositional variable v_j is associated with a unary predicate P_j, and each $1 \leq i \leq n$ is associated with a binary relation R_i which is used in the translation of $\Box_i$- and $\Diamond_i$-formulae. The translation function f_x takes as second parameter a first-order variable x; this is because $\mathcal{K}_n$-formulae are mapped to first-order logic formulae with one free

variable. Thus, when applying the translation function ST to a $\mathcal{K}_n$-formula φ and variable x, we obtain a first-order formula $ST(\varphi, x)$ which has x as its unique free variable. We demonstrate the translation with an example.

Example 2.7.1.

$$\begin{aligned}
&ST(\Box_1(\neg v_1 \wedge \Diamond_2 v_3), x)\\
=&\forall y(R_1(x,y) \rightarrow ST(\neg v_1 \wedge \Diamond_2 v_3, y))\\
=&\forall y(R_1(x,y) \rightarrow ST(\neg v_1, y) \wedge ST(\Diamond_2 v_3, y))\\
=&\forall y(R_1(x,y) \rightarrow \neg P_1(y) \wedge ST(\Diamond_2 v_3, y))\\
=&\forall y(R_1(x,y) \rightarrow \neg P_1(y) \wedge \exists z(R_2(y,z) \wedge ST(v_3, z)))\\
=&\forall y(R_1(x,y) \rightarrow \neg P_1(y) \wedge \exists z(R_2(y,z) \wedge P_3(z)))
\end{aligned}$$

The following theorem shows that the translation ST is satisfiability-preserving.

Theorem 2.7.2.
Let φ be a $\mathcal{K}_n$-formula, let $\mathfrak{M} = \langle \mathcal{W}, \{\mathcal{R}_i\}_{i=1}^n, v\rangle$ be a $\mathcal{K}_n$-model, and let $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}\rangle$ be the first-order logic model defined as follows: $\Delta^{\mathcal{I}} = \mathcal{W}$, $P_j^{\mathcal{I}} = v(v_j)$, and $R_i^{\mathcal{I}} = \mathcal{R}_i$. Finally, consider some $w \in \mathcal{W}$, and let s be a variable assignment which maps the variable x to w. Then we have:

$$\mathfrak{M}, w \models \varphi \text{ if and only if } \mathcal{I}, s \models ST(\varphi, x)$$

Because of Theorem 2.7.2, results concerning first-order logic can be transferred to $\mathcal{K}_n$. This allows us for instance to derive that $\mathcal{K}_n$ has the compactness property:

Theorem 2.7.3.
Let Σ be a set of $\mathcal{K}_n$-formulae. If every finite subset of Σ is satisfiable, then Σ is also satisfiable.

Proof. Direct consequence of the analogous result for first-order logic (cf. Theorem 1.3.22 of [CK90]) and the embedding of $\mathcal{K}_n$ in first-order logic (Theorem 2.7.2). □

2.8 Relation to Description Logics

Like modal logics, description logics are a family of knowledge representation languages which offer more expressivity than propositional logic but better computational properties than first-order logic. In this section, we start by providing a general overview of description logics, before moving on to discuss two specific description logics ($\mathcal{ALC}$ and $\mathcal{ALE}$) and their relation to the modal logic $\mathcal{K}_n$.

2.8.1 A short introduction to description logics

The basic building blocks of all description logics are atomic concepts and atomic roles, which correspond respectively to unary and binary predicates. We might for example have atomic concepts $Female$ and $Teacher$ and atomic roles $HasChild$ and $IsFriendOf$. More complex concepts and roles can be built from the set of atomic concepts and roles by using concept and role constructors. So for instance, given two concepts $Female$ and $Teacher$, and the constructor conjunction ($\sqcap$), we can form the complex concept $Female \sqcap Teacher$, which describes the set of female teachers. The set of constructors available depends on the description logic in question.

Description logic knowledge bases are composed of two parts, an ABox and a TBox. The ABox makes statements about the properties of specific individuals and relationships between individuals. It is composed of a finite set of assertions of the following forms:

$$C(a) \qquad\qquad R(a, b)$$

where a and b are named individuals, C is a concept expression, and R a role expression. The assertion $C(a)$ states that a is an instance of the concept C, and the assertion $R(a, b)$ indicates that a stands in the relationship R to b. Typical ABox axioms might be $Teacher \sqcup Doctor(mary)$ and $HasChild(mary, john)$.

The TBox is composed of a set of terminological axioms which allow us to describe the relationship between different concepts [4]. Terminological axioms have one of the following two forms:

$$C \equiv D \qquad\qquad C \sqsubseteq D$$

where C and D are both concept expressions. The first axiom states that the concepts C and D describe the same set of individuals, whereas the second states that the concept D is more general than C. Some examples of TBox axioms are $Mother \equiv Female \sqcap \exists HasChild.\top$, $Parent \equiv Mother \sqcup Father$, and $Cat \sqsubseteq Animal$.

The meaning of concept expressions, ABox assertions, and TBox axioms is given via a model-theoretic semantics which is quite similar to that of first-order logic. An interpretation $\mathcal{I}$ is defined to be a pair $\langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$, where $\Delta^{\mathcal{I}}$ is a non-empty set and $\cdot^{\mathcal{I}}$ is a function mapping each atomic concept A to a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, each atomic role $R \in \mathcal{R}$ to a relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and each individual name a to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ of the universe. The function $\cdot^{\mathcal{I}}$ is straightforwardly extended to

4. Some description logics also permit role axioms.

handle complex concept and role expressions. For example, conjunction of concepts is interpreted as intersection of the sets corresponding to the concepts. Thus, to every concept expression C is associated a subset $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and to every role expression S is associated a relation $S^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

A concept C is said to be *satisfiable* if there is some interpretation $\mathcal{I}$ for which $C^{\mathcal{I}} \neq \emptyset$. If there is no such model, then C is said to be *unsatisfiable*, and we write $\models C \sqsubseteq \bot$. We say that a concept C is *subsumed* by D (or that D *subsumes* C), written $\models C \sqsubseteq D$, if for every model $\mathcal{I}$ we have $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. An ABox assertion $C(a)$ is said to hold in an interpretation $\mathcal{I}$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$. An assertion $R(a, b)$ is verified by $\mathcal{I}$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$. An ABox $\mathcal{A}$ is satisfied by an interpretation $\mathcal{I}$ if every assertion in $\mathcal{A}$ holds in $\mathcal{I}$. An ABox $\mathcal{A}_1$ entails another ABox $\mathcal{A}_2$ if every interpretation which satisfies $\mathcal{A}_1$ also satisfies $\mathcal{A}_2$. A TBox axiom $C \sqsubseteq D$ (resp. $C \equiv D$) is satisfied by a model $\mathcal{I}$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ (resp. $C^{\mathcal{I}} = D^{\mathcal{I}}$). A TBox is satisfied in $\mathcal{I}$ if all of its axioms are satisfied in $\mathcal{I}$. A TBox $\mathcal{T}_1$ entails another TBox $\mathcal{T}_2$ just in the case that every model of $\mathcal{T}_1$ is also an model of $\mathcal{T}_2$.

Some typical description logic reasoning tasks are:

- *Concept satisfiability*: Is the concept C satisfiable?
- *Subsumption*: Is the concept C subsumed by the concept D?
- *Abox entailment*: Does the ABox $\mathcal{A}_1$ entail the ABox $\mathcal{A}_2$?
- *TBox entailment*: Does the TBox $\mathcal{T}_1$ entail the TBox $\mathcal{T}_2$?

2.8.2 The description logic $\mathcal{ALC}$

The description logic that will be of most interest to us in this thesis is $\mathcal{ALC}$. Concepts expressions in $\mathcal{ALC}$ are built up from atomic concepts and roles using the following constructors: negation ($\neg$), conjunction ($\sqcap$), disjunction ($\sqcup$), universal role restriction ($\forall$), and existential role restriction ($\exists$). Formally, the syntax of concept expressions is defined recursively as follows:

$$C ::= \top \mid \bot \mid A \mid \neg C \mid C \sqcap C \mid C \sqcup C \mid \forall R.C \mid \exists R.C$$

where A is an atomic concept and R an atomic role. A typical $\mathcal{ALC}$ expression might be

$$Male \sqcap Teacher \sqcap \exists hasChild.\top \sqcap \forall hasChild.(Doctor \sqcup Teacher)$$

which describes the set of male teachers that are fathers having only doctors and teachers as children.

$$
\begin{aligned}
f(\top) &= \top \\
f(\bot) &= \bot \\
f(a_j) &= A_j \\
f(\neg\varphi) &= \neg f(\varphi) \\
f(\varphi \wedge \psi) &= f(\varphi) \sqcap f(\psi) \\
f(\varphi \vee \psi) &= f(\varphi) \sqcup f(\psi) \\
f(\Diamond_i \varphi) &= \exists R_i.f(\varphi) \\
f(\Box_i \varphi) &= \forall R_i.f(\varphi)
\end{aligned}
$$

(a) From $\mathcal{K}_n$ to $\mathcal{ALC}$.

$$
\begin{aligned}
g(\top) &= \top \\
g(\bot) &= \bot \\
g(A_j) &= a_j \\
g(\neg C) &= \neg g(C) \\
g(C \sqcap D) &= g(C) \wedge g(D) \\
g(C \sqcup D) &= g(C) \vee g(D) \\
g(\exists R_i.C) &= \Diamond_i g(C) \\
g(\forall R_i.C) &= \Box_i g(C)
\end{aligned}
$$

(b) From $\mathcal{ALC}$ to $\mathcal{K}_n$.

Figure 2.4: Mapping between $\mathcal{K}_n$ formulae and $\mathcal{ALC}$ concept expressions.

The semantics of the different $\mathcal{ALC}$ constructors is defined as follows:

$$
\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} &= \emptyset \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \forall b.\, (a,b) \in R^{\mathcal{I}} \Rightarrow b \in C^{\mathcal{I}}\} \\
(\exists R.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \exists b.\, (a,b) \in R^{\mathcal{I}} \text{ and } b \in C^{\mathcal{I}}\}
\end{aligned}
$$

Correspondence between $\mathcal{K}_n$ and $\mathcal{ALC}$

In Figure 2.4, we define two functions, one mapping $\mathcal{K}_n$ formulae to $\mathcal{ALC}$ concept expressions, and the second mapping $\mathcal{ALC}$ concept expressions to $\mathcal{K}_n$ formulae. The mappings are quite straightforward: atomic concepts are associated with propositional variables, role restrictions are associated with modal operators, and the boolean concept constructors are mapped to the corresponding Boolean connectives.

Theorem 2.8.1 ([Sch91]).
Let f and g be as defined in Figure 2.4.

1. *For $\mathcal{K}_n$-formula φ and ψ: $\varphi \models \psi$ if and only if $\models f(\varphi) \sqsubseteq f(\psi)$*
2. *For $\mathcal{ALC}$ concepts C and D: $\models C \sqsubseteq D$ if and only if $g(C) \models g(D)$*

Because of Theorem 2.8.1, results concerning $\mathcal{K}_n$-formulae (with respect to the local consequence relation) can be transferred to $\mathcal{ALC}$ concept expressions, and vice-versa. *This means in particular that all of the results that we will establish in the following chapters for $\mathcal{K}_n$-formulae apply equally well to concept expressions in $\mathcal{ALC}$.*

Remark 2.8.2.
Entailment between $\mathcal{ALC}$ TBoxes can also be rephrased in terms of $\mathcal{K}_n$ formulae, but for this, the global consequence relation is required. ABoxes, on the other hand, cannot be represented in the logic $\mathcal{K}_n$ since $\mathcal{K}_n$ (like most modal logics) does not provide any means of referring to particular worlds.

2.8.3 The description logic $\mathcal{ALE}$

In later chapters, we will also make reference to the description logic $\mathcal{ALE}$, which is obtained from $\mathcal{ALC}$ by disallowing disjunction and general negation of concepts. Formally, the syntax of $\mathcal{ALE}$ concept expressions is defined recursively as follows:

$$C ::= \top \mid \bot \mid A \mid \neg A \mid C \sqcap C \mid \forall R.C \mid \exists R.C$$

Using the mappings between $\mathcal{K}_n$ and $\mathcal{ALC}$ from the previous subsection, we see that $\mathcal{ALE}$ expressions correspond precisely to the set of $\mathcal{K}_n$ formulae which are in negation normal form and do not contain any disjunction symbols.

The reduced expressiveness of $\mathcal{ALE}$ compared to $\mathcal{ALC}$ is rewarded by a drop in the complexity of reasoning: both the unsatisfiability and subsumption tasks [5] for $\mathcal{ALE}$ concept expressions can be accomplished in non-deterministic polynomial time, whereas the corresponding problems for $\mathcal{ALC}$ are PSPACE-complete.

Theorem 2.8.3 ([SSS91], [DLN$^+$92]).
Unsatisfiability and subsumption of $\mathcal{ALE}$ concept expressions are both in NP.

Proof. We give a proof for unsatisfiability, and refer the reader to [DLN$^+$92] for subsumption. Consider the following non-deterministic procedure for deciding the unsatisfiability of an $\mathcal{ALE}$ concept C:

(1) Guess a (possibly empty) sequence $S_1, ..., S_n$ of subconcepts [6] of C such that $n \leq \delta(C)$ and each S_i is of the form $\exists R.E$. Set D equal to C.

5. For description logics like $\mathcal{ALC}$ which allow for full negation, concept satisfiability and subsumption can be reduced to one another, but for less expressive logics, these tasks can have different complexities.

6. The notion of subconcept is defined analogously to that of subformula. Likewise, the size and depth of a concept are defined in the same manner as for formulae.

(2) For $i = 1$ to n
    If $S_i = \exists R.E$ is a conjunct of D,
        Set $D = E \sqcap (\sqcap_{F \in \mathcal{F}} F)$, where $\mathcal{F} = \{F \mid \forall R.F \text{ is a conjunct of } D\}$
    Else, return **no**
(3) Return **yes** if D has a conjunct $\bot$ or a pair of conjuncts $A, \neg A$, else return **no**.

This procedure clearly runs in non-deterministic polynomial time since in Step 1 we guess at most $n \leq |C|$ concepts each with size at most $|C|$, and there are at most $n \leq |C|$ iterations of the for loop in Step 2, each iteration taking only a polynomial amount of time.

We now show that the above procedure outputs **yes** just in the case the input concept is unsatisfiable. For the first direction, suppose the output on C is **yes**, and let $S_1, ..., S_n$ be the sequence of subconcepts guessed in Step 1. It can be easily shown by induction that the concept D at the beginning of Step 3 must satisfy $\models C \sqsubseteq (\exists R)^n D$. Moreover, we also know that D must have a conjunct $\bot$ or a pair of conjuncts A and $\neg A$, since the output on C is **yes**. It follows that $\models C \sqsubseteq (\exists R)^n \bot$, and hence $\models C \sqsubseteq \bot$. For the other direction, suppose C is unsatisfiable. We set $D_1 = C$, and we construct a sequence of concepts $S_1, ..., S_m$ in the following manner. If at stage i, the concept D_i has a conjunct $\bot$ or a pair of conjuncts A and $\neg A$, we return the empty sequence. Otherwise, the concept D_i must possess conjuncts $\exists R.E$, $\forall R.F_1$, ..., $\forall R.F_m$ such that $E \sqcap F_1 \sqcap ... \sqcap F_m$ is unsatisfiable. We set $S_i = \exists R.E$ and set $D_{i+1} = E \sqcap F_1 \sqcap ... \sqcap F_m$. We remark that there can be at most $\delta(C)$ elements in the constructed sequence since the depth of D_{i+1} is at least one less than the depth of D_i. We also remark that if the constructed sequence has n elements, then the concept D_{n+1} has either a conjunct $\bot$ or a pair of conjuncts $A, \neg A$. Moreover, it is easily verified that if the sequence of subconcepts we have constructed is guessed in Step 1, then the concept examined in Step 3 is precisely the concept D_{n+1}. It follows that there exists a sequence which leads to an output of **yes** on input C. □

NP-hardness of the unsatisfiability and subsumption tasks can also be demonstrated.

Theorem 2.8.4 ([DLN$^+$92]).
For $\mathcal{ALE}$ concept expressions, unsatisfiability and subsumption are both NP*-hard.*

Proof. The original proof in [DLN$^+$92] uses a reduction from the NP-complete problem One-in-three 3SAT, but here we outline another reduction from the exact cover problem which was presented in [Don03]. The exact cover problem (cf. [GJ79]) is the following: given a set $\mathcal{U} = \{u_1, ..., u_n\}$ and a set $\mathcal{S} = \{S_1, ..., S_m\}$ of subsets

of $\mathcal{U}$, determine whether there exists an exact cover, that is, a subset $\{S_{j_1}, ..., S_{j_q}\}$ of $\mathcal{S}$ such that $S_{j_h} \cap S_{j_k} = \emptyset$ for $h \neq k$ and $\bigcup_{k=1}^{q} S_{j_k} = \mathcal{U}$. We will show that $\mathcal{U}, \mathcal{S}$ has an exact cover if and only if the $\mathcal{ALE}$ concept $C_{\mathcal{U},\mathcal{S}}$ pictured in Figure 2.5 is unsatisfiable.

$$C_{\mathcal{U},\mathcal{S}} = D_{1,1} \sqcap ... \sqcap D_{1,m} \sqcap E$$

where the $D_{i,j}$ are defined inductively as follows

$$D_{i,j} = \begin{cases} \exists R.D_{i+1,j}, \text{if either } i \leq n, u_i \in S_j, \text{ or } i > n \text{ and } u_{i-n} \in S_j \\ \forall R.D_{i+1,j}, \text{if either } i \leq n, u_i \notin S_j, \text{ or } i > n \text{ and } u_{i-n} \notin S_j \end{cases}$$

for $i \in \{1, ..., 2n\}$ and $D_{2n+1,j} = \top$, and $E = \underbrace{\forall R....\forall R.}_{2n} \bot$.

Figure 2.5: The concept $C_{\mathcal{U},\mathcal{S}}$ which codes an instance $\mathcal{U} = \{u_1, ..., u_n\}$, $\mathcal{S} = \{S_1, ..., S_m\}$ of the exact cover problem.

The first direction ($\mathcal{U}, \mathcal{S}$ has an exact cover $\Rightarrow C_{\mathcal{U},\mathcal{S}}$ is unsatisfiable) is rather straightforward, so we concentrate on the second part of the equivalence. Suppose then that the concept $C_{\mathcal{U},\mathcal{S}}$ is unsatisfiable. It follows that $\models D_{1,1} \sqcap ... \sqcap D_{1,m} \sqsubseteq (\exists R)^{2n} \top$. We partition $\{1, ..., m\}$ into two sets: a first set $\mathcal{J}_1^{\exists}$ containing those indices j for which $D_{1,j} = \exists R.D_{2,j}$, and a set $\mathcal{J}_1^{\forall}$ containing those indices j for which $D_{1,j} = \forall R.D_{2,j}$. We next define inductively a sequence of integers $h_1, ..., h_{2n}$ and sequences of sets $\mathcal{J}_2^{\exists}, ..., \mathcal{J}_{2n}^{\exists}$ and $\mathcal{J}_2^{\forall}, ..., \mathcal{J}_{2n}^{\forall}$ in the following manner:

- h_i is an element of $\mathcal{J}_i^{\exists}$ such that $\models (\sqcap_{j \in \mathcal{J}_i^{\forall}} D_{i+1,j}) \sqcap D_{i+1,h_i} \sqsubseteq (\exists R)^{2n-i} \top$
- $\mathcal{J}_i^{\exists} = \{j \mid j \in \mathcal{J}_{i-1}^{\forall} \cup \{h_{i-1}\} \text{ and } D_{i,j} = \exists R.D_{i+1,j}\}$
- $\mathcal{J}_i^{\forall} = \{j \mid j \in \mathcal{J}_{i-1}^{\forall} \cup \{h_{i-1}\} \text{ and } D_{i,j} = \forall R.D_{i+1,j}\}$

This definition is well-founded since the initial sets $\mathcal{J}_1^{\exists}$ and $\mathcal{J}_1^{\exists}$ have already been defined above. Moreover, the fact that h_i is chosen so that $\models (\sqcap_{j \in \mathcal{J}_i^{\forall}} D_{i+1,j}) \sqcap D_{i+1,h_i} \sqsubseteq (\exists R)^{2n-i} \top$ guarantees the existence of an element h_{i+1} of $\mathcal{J}_{i+1}^{\exists}$ with the required properties. We remark that by construction for every $1 \leq i \leq 2n-1$ we have:

(a) $\mathcal{J}_{i+1}^{\exists} \cup \mathcal{J}_{i+1}^{\forall} \subseteq \mathcal{J}_i^{\exists} \cup \mathcal{J}_i^{\forall}$
(b) $|\mathcal{J}_i^{\exists} \cap (\mathcal{J}_{i+1}^{\exists} \cup \mathcal{J}_{i+1}^{\forall})| = 1$

We intend to show that $\Sigma = \{S_{h_{n+1}}, ..., S_{h_{2n}}\}$ is an exact cover for $\mathcal{U}, \mathcal{S}$. We first remark that because of the way that h_i are defined, for each $1 \leq i \leq n$, we have $D_{n+i,h_{n+i}} = \exists R.D_{n+i+1,h_{n+i}}$, which means that each element $u_i \in \mathcal{U}$ belongs to

some set in Σ (namely the set $S_{h_{n+i}}$). It remains to be shown that the sets in Σ are pairwise disjoint. Suppose for a contradiction that some u_i appears in two distinct sets $S_{h_{n+f}}$ and $S_{h_{n+g}}$ in Σ. That means that $D_{i,h_{n+f}} = \exists R.D_{i+1,h_{n+f}}$ and $D_{i,h_{n+g}} = \exists R.D_{i+1,h_{n+g}}$. We also know that $h_{n+f} \in \mathcal{J}^{\exists}_{n+f}$ and $h_{n+g} \in \mathcal{J}^{\exists}_{n+g}$, and hence $h_{n+f} \in \mathcal{J}^{\exists}_{i}$ and $h_{n+g} \in \mathcal{J}^{\exists}_{i}$ (by (a)). It follows then from (b) that either $h_{n+f} \notin \mathcal{J}^{\exists}_{i+1} \cup \mathcal{J}^{\forall}_{i+1}$ or $h_{n+g} \notin \mathcal{J}^{\exists}_{i+1} \cup \mathcal{J}^{\forall}_{i+1}$. But then using (a), we find that either $h_{n+f} \notin \mathcal{J}^{\exists}_{n+f}$ or $h_{n+g} \notin \mathcal{J}^{\exists}_{n+g}$, which is a contradiction. □

Note that the concept $C_{\mathcal{U},\mathcal{S}}$ used in the reduction in the preceding proof has a very simple syntax (conjunction of strings of role restrictions followed by atomic literals). We will make use of this fact in later chapters.

3

Prime Implicates and Prime Implicants in $\mathcal{K}_n$

The purpose of this chapter is to select a suitable definition of prime implicates and prime implicants for the logic $\mathcal{K}_n$. The first half of the chapter will be concerned with the generalization of the notions of clauses and terms to $\mathcal{K}_n$. As there is no obvious definition, we will enumerate a list of syntactic, semantic, and complexity-theoretic properties of propositional clauses and terms, which we will then use to compare the different candidate definitions. In the second half of the chapter, we will consider the different definitions of clauses and terms in light of the notions of prime implicate and prime implicant they induce. Once again, we will list some basic properties from the propositional case that we would like to satisfy, and we will see how the different definitions measure up.

3.1 Defining Clauses and Terms in $\mathcal{K}_n$

As we have seen in Chapter 1, the notions of prime implicates and prime implicants are straightforwardly defined using the notions of clauses and terms. Thus, if we aim to provide suitable definitions of prime implicates and prime implicants for $\mathcal{K}_n$, a logical first step is to come up with an appropriate definition of clauses and terms in $\mathcal{K}_n$. Unfortunately, whereas clauses and terms are standard notions in both propositional and first-order logic [1], there is no generally accepted definition

1. One might wonder why we don't simply translate our formulae in $\mathcal{K}_n$ into first-order formulae and then put them into clausal form. The reason is simple: we are looking to define clauses and terms *within* the language of $\mathcal{K}_n$, and the clauses we obtain on passing by first-order logic are

of clauses and terms in $\mathcal{K}_n$. Indeed, a couple of different notions of clauses and/or terms for $\mathcal{K}_n$ have been proposed in the literature for various purposes.

Instead of blindly picking a definition and hoping that it is appropriate, we prefer to list a number of characteristics of literals, clauses, and terms in propositional logic, which will provide us with a principled means of comparing different candidate definitions. Each of the properties below describes something of what it is to be a literal, clause, or term in propositional logic. Although our list cannot be considered exhaustive, we do believe that it covers the principal syntactic, semantic, and complexity-theoretic properties of the propositional definition.

P1 Literals, clauses, and terms are in negation normal form.

P2 Clauses do not contain $\wedge$, terms do not contain $\vee$, and literals contain neither $\wedge$ nor $\vee$.

P3 Clauses (resp. terms) are disjunctions (resp. conjunctions) of literals.

P4 The negation of a literal is equivalent to another literal. Negations of clauses (resp. terms) are equivalent to terms (resp. clauses).

P5 Every formula is equivalent to a finite conjunction of clauses. Likewise, every formula is equivalent to a finite disjunction of terms.

P6 The task of deciding whether a given formula is a literal, term, or clause can be accomplished in polynomial-time.

P7 The task of deciding whether a clause (resp. term) entails another clause (resp. term) can be accomplished in polynomial-time.

3.1.1 Impossibility result

A natural question is whether there exist definitions of literals, clauses, and terms for $\mathcal{K}_n$ satisfying all of the aforementioned properties. Unfortunately, the following impossibility result shows this not to be the case.

Theorem 3.1.1.
*Any definition of literals, clause, and terms for $\mathcal{K}$ that satisfies properties **P1** and **P2** cannot satisfy **P5**.*

Proof. Let us define clauses (resp. terms) to be the set of formulae in NNF which do not contain $\wedge$ (resp. $\vee$). This is clearly the most expressive definition of clauses and terms satisfying both **P1** and **P2**, so to show the result, it suffices to show that

generally not expressible in $\mathcal{K}_n$. Moreover, if we were to define clauses in $\mathcal{K}_n$ as those first-order clauses which are representable in $\mathcal{K}_n$, we would obtain a set of clauses containing no $\Diamond$ modalities, thereby losing much of the expressivity of $\mathcal{K}_n$.

this definition does not satisfy **P5**.

Suppose for a contradiction that this definition does satisfy **P5**. Then there must exist clauses $\lambda_1, ..., \lambda_n$ such that $\Diamond(a \wedge b) \equiv \lambda_1 \wedge ... \wedge \lambda_n$. Each of the clauses λ_i is a disjunction $l_{i,1} \vee .... \vee l_{i,p_i}$. By distributing $\wedge$ over $\vee$, we obtain the following:

$$\Diamond(a \wedge b) \equiv \bigvee_{(j_1,...,j_n) \in \{1,...,p_1\} \times ... \times \{1,...,p_n\}} \bigwedge_{i=1}^{n} l_{i,j_i}$$

from which we can infer that for each $(j_1, ..., j_n) \in \{1, ..., p_1\} \times ... \times \{1, ..., p_n\}$ we have

$$\bigwedge_{i=1}^{n} l_{i,j_i} \models \Diamond(a \wedge b)$$

Consider some $(j_1, ..., j_n)$ such that $\bigwedge_{i=1}^{n} l_{i,j_i}$ is consistent (there must be at least one such tuple, otherwise we would have $\Diamond(a \wedge b) \equiv \bot$). The formulae l_{i,j_i} are either propositional literals or formulae of the form $\Box\kappa$ or $\Diamond\kappa$ for some clause κ. It follows that $\bigwedge_{i=1}^{n} l_{i,j_i}$ must have the following form:

$$\gamma_1 \wedge ... \wedge \gamma_k \wedge \Diamond\psi_1 \wedge ... \wedge \Diamond\psi_m \wedge \Box\chi_1 \wedge ... \wedge \Box\chi_n$$

where γ_1, ..., γ_k are propositional literals and ψ_1, ..., ψ_m, χ_1, ..., χ_n are clauses with respect to the definition we have chosen. As we know that $\bigwedge_{i=1}^{n} l_{i,j_i} \models \Diamond(a \wedge b)$ and $\bigwedge_{i=1}^{n} l_{i,j_i} \not\models \bot$, by Theorem 2.3.1, there must be some $\Diamond\psi_q$ such that

$$\Diamond\psi_q \wedge \Box\chi_1 \wedge ... \wedge \Box\chi_n \models \Diamond(a \wedge b)$$

We now show that $\Diamond\psi_q \not\models \Diamond(a \wedge b)$ (and hence that $\not\models \chi_1 \wedge ... \wedge \chi_n$). Suppose for a contradiction that this is not the case. Then we must have $\psi_q \models a$ and $\psi_q \models b$. But by Theorem 2.3.1, every disjunct of ψ_q (which we recall is a clause w.r.t. our supposed definition) must either be unsatisfiable or equal to both a and b. As the latter is impossible, it follows that $\psi_q \models \bot$, which is a contradiction since we assumed that $\bigwedge_{i=1}^{n} l_{i,j_i}$ is satisfiable. It follows then that in order to get $\Diamond\psi_q \wedge \Box\chi_1 \wedge ... \wedge \Box\chi_n \models \Diamond(a \wedge b)$, there must be some χ_r which is not a tautology.

Now let us consider the formula

$$\tau = \bigvee_{\{(j_1,...,j_n) | \bigwedge_{i=1}^{n} l_{i,j_i} \not\equiv \bot\}} \Box\chi_{j_1,...,j_n}$$

where $\Box\chi_{j_1,...,j_n}$ is a non-tautological $\Box$-formula appearing in $\bigwedge_{i=1}^{n} l_{i,j_i}$ (we have just shown that such a formula must exist). Clearly it must be the case that

$$\bigvee_{(j_1,...,j_n) \in \{1,...,p_1\} \times ... \times \{1,...,p_n\}} \bigwedge_{i=1}^{n} l_{i,j_i} \models \tau$$

from which we get

$$\Diamond(a \wedge b) \models \tau$$

But according to Theorem 2.3.2, a satisfiable $\Diamond$-formula cannot imply a disjunction of $\Box$-formulae unless that disjunction is a tautology, so we must have $\models \tau$. However, this is impossible since it would imply (Theorem 2.3.1) that there is some $\chi_{j_1,...,j_n}$ which is a tautology, contradicting our earlier assumption to the contrary. We can thus conclude that there is no set of clauses $\lambda_1, ..., \lambda_n$ with respect to our selected definition such that $\Diamond(a \wedge b) \equiv \lambda_1 \wedge ... \wedge \lambda_n$, and hence that any definition which satisfies **P1** and **P2** cannot satisfy **P5**. □

The proof of Theorem 3.1.1 only makes use of the fact that $\wedge$ does not distribute over $\Diamond$ and $\vee$ does not distribute over $\Box$, which means that our impossibility result holds equally well for most standard modal and description logics.

3.1.2 Analysis of candidate definitions

We will now consider a variety of possible definitions and evaluate them with respect to the above criteria. Note that in what follows, we let i range over the integers between 1 and n, a range over the set of propositional variables, and L, C, and T range over the sets of literals, clauses, and terms respectively.

Definition D1

The first definition that we will consider is that proposed in [CP95] in the context of abduction. The authors define terms to be the formulae which can be constructed from the propositional literals using only $\wedge$ and the modal operators. Modal clauses and literals are not used in the paper but can be defined analogously, yielding the following definition:

$$\textbf{D1} \qquad \begin{aligned} L &::= \top \mid \bot \mid a \mid \neg a \mid \Box_i L \mid \Diamond_i L \\ C &::= \top \mid \bot \mid a \mid \neg a \mid \Box_i C \mid \Diamond_i C \mid C \vee C \\ T &::= \top \mid \bot \mid a \mid \neg a \mid \Box_i T \mid \Diamond_i T \mid T \wedge T \end{aligned}$$

It is easy to see by inspection that this definition satisfies properties **P1** and **P2**. It is also easy to see that property **P6** is satisfied since **D1** is a context-free grammar, and it is well-known that the membership problem for context-free grammars can be solved in polynomial time (cf. [You67]). Property **P4** can also be shown to hold:

Lemma 3.1.2.
*Definition **D1** satisfies property **P4**.*

Proof. We can show by induction on the structural complexity of formulae that the function **Nnf** maps negations of literals to literals, negations of clauses to terms, and negations of terms to clauses. As the proof is straightforward and rather tedious, we will only give the proof for the case of clauses.

The base case is when the input to **Nnf** is the negation of a propositional literal. The statement holds in this case since **Nnf** maps $\neg a$ to $\neg a$ and $\neg(\neg a)$ to a, and both a and $\neg a$ are terms with respect to definition **D1**.

Next let us suppose that the statement holds for clauses λ_1 and λ_2, and let $\bar{\lambda_1}$ and $\bar{\lambda_2}$ respectively be the formulae output by the function **Nnf** on inputs λ_1 and λ_2. We now want to show that the result holds for more complex clauses built from λ_1 and λ_2. If **Nnf** is called on input $\neg\Box_i \lambda_1$, then because of the induction hypothesis, we know that the output will be the formula $\Diamond_i \bar{\lambda_1}$, which is a term with respect to **D1**. If instead the input to **Nnf** is of the form $\neg\Diamond_i \lambda_1$, then **Nnf** will output the **D1**-term $\Box_i \bar{\lambda_1}$. Finally, if the input to **Nnf** is the clause $\lambda_1 \vee \lambda_2$, then the output will be the term $\bar{\lambda_1} \wedge \bar{\lambda_2}$. □

The remaining properties are not satisfied by definition **D1**. Property **P3** is falsified since there are clauses that are not disjunctions of literals – take for instance the clause $\Box(a \vee b)$. Property **P5** cannot hold because of our impossibility result (Theorem 3.1.1). At first glance, it may seem that entailment between clauses or terms of **D1** could be accomplished in polynomial time (property **P7**), but this turns out not to be the case. In fact, we can show this problem to be NP-complete. The proof exploits the following correspondence between terms of **D1** and concept expressions in the description logic $\mathcal{ALE}$.

Lemma 3.1.3.

1. *The function f in Figure 2.4a maps **D1**-terms into $\mathcal{ALE}$ concept expressions.*
2. *The function g in Figure 2.4b maps $\mathcal{ALE}$ concept expressions into **D1**-terms.*

Proof Sketch. Straightforward structural induction proof. □

Lemma 3.1.4.
Entailment between terms or clauses is NP*-complete for definition **D1**.*

Proof. It follows from Theorem 2.8.1 that $\tau_1 \models \tau_2$ if and only if $\models f(\tau_1) \sqsubseteq f(\tau_2)$. We also know by Lemma 3.1.3 that if τ_1 and τ_2 are terms with respect to **D1**, then $f(\tau_1)$ and $f(\tau_2)$ must be concept expressions in $\mathcal{ALE}$. This means that we can reduce entailment between terms with respect to **D1** to subsumption between $\mathcal{ALE}$ concepts. As concept subsumption in $\mathcal{ALE}$ is known to belong to the class

NP (Theorem 2.8.3), it follows that entailment between **D1**-terms must also belong to NP.

For NP-hardness, we use the function g from Figure 2.4b to map the concept $C_{\mathcal{U},\mathcal{S}}$ from Figure 2.5 into a $\mathcal{K}_n$ formula $\varphi_{\mathcal{U},\mathcal{S}}$. We know from Lemma 3.1.3 that $\varphi_{\mathcal{U},\mathcal{S}}$ is a term with respect to **D1**, and from Theorem 2.8.1 that $\varphi_{\mathcal{U},\mathcal{S}}$ is satisfiable just in the case that the concept $C_{\mathcal{U},\mathcal{S}}$ is. As it was shown in [Don03] that $\mathcal{U},\mathcal{S}$ has an exact cover if and only if $C_{\mathcal{U},\mathcal{S}}$ is unsatisfiable, it follows that $\mathcal{U},\mathcal{S}$ has an exact cover if and only if $\varphi_{\mathcal{U},\mathcal{S}}$ is unsatisfiable. But a term is unsatisfiable just in the case that it entails the term $\bot$. This means the XC decision problem can be polynomially-reduced to entailment between **D1**-terms, making the latter problem NP-hard and hence NP-complete.

In order to show the NP-completeness of clausal entailment, we remark that for definition **D1**, the function **Nnf** transforms negations of clauses into terms and negations of terms into clauses (cf. proof of Lemma 3.1.2). This means that we can test whether a clause λ entails a clause λ' by testing whether the term $\mathbf{Nnf}(\neg\lambda')$ entails the term $\mathbf{Nnf}(\neg\lambda)$. Likewise, we can test whether a term κ entails another term κ' by testing whether the clause $\mathbf{Nnf}(\neg\kappa')$ entails the clause $\mathbf{Nnf}(\neg\kappa)$. As the NNF transformation is polynomial, it follows that entailment between clauses is exactly as difficult as entailment between terms, so clausal entailment is NP-complete. □

Remark 3.1.5.
For the proof of Lemma 3.1.4, we made use of the fact that $\mathcal{K}_n$ contains the symbols $\top$ and $\bot$. If we choose not to include these symbols in the language, then we need to modify the NP-hardness proof, since the formula $\varphi_{\mathcal{U},\mathcal{S}}$ used in the reduction contains both $\top$ and $\bot$. The modification is straightforward: we replace occurrences of $\top$ in $\varphi_{\mathcal{U},\mathcal{S}}$ by a and replace $\bot$ by $\neg a$. The resulting formula $\varphi'_{\mathcal{U},\mathcal{S}}$ is a **D1**-term which is satisfiable whenever $\varphi_{\mathcal{U},\mathcal{S}}$ is, so we can use $\varphi'_{\mathcal{U},\mathcal{S}}$ in place of $\varphi_{\mathcal{U},\mathcal{S}}$ in the proof.

Theorem 3.1.6.
*Definition **D1** satisfies properties **P1**, **P2**, **P4**, and **P6**, and falsifies properties **P3**, **P5**, and **P7**.*

Proof. Follows from Lemmas 3.1.2 and 3.1.4 and the preceding discussion. □

Definition D2

If we take the notion of literals from **D1** and use it to construct the set of clauses and terms, we obtain the following definition:

$$
\begin{array}{ll}
 & L ::= \top \,|\, \bot \,|\, a \,|\, \neg a \,|\, \Box_i L \,|\, \Diamond_i L \\
\textbf{D2} & C \ ::= L \,|\, C \vee C \\
 & T ::= L \,|\, T \wedge T
\end{array}
$$

It can be easily verified that definition **D2** satisfies properties **P1**-**P3**. For **P6**, we again use the fact that **D2** defines a context-free grammar. The proof of **P4** is quite straightforward and similar that for definition **D1**, so will omit it here. As for property **P5**, we can either use Theorem 3.1.1, or we can simply remark that definition **D2** is even less expressive than **D1**, and we have already shown **D1** to falsify **P5**. The reduced expressiveness of **D2** does not however improve its computational complexity: property **P7** is still not satisfied as we can show that entailment between clauses or terms is NP-complete using exactly the same reduction as was used for definition **D1**. The fact that even an extremely inexpressive definition like **D2** does not allow for polynomial entailment between clauses and terms suggests that property **P7** cannot be satisfied by any reasonable definition of clauses and terms for $\mathcal{K}_n$.

Theorem 3.1.7.
*Definition **D2** satisfies properties **P1**- **P4** and **P6** and falsifies **P5** and **P7**.*

Proof. It is easy to see that every **D2**-clause is also a **D1**-clause, and likewise every **D2**-term is also a **D1**-term. It follows then that for definition **D2** entailment between clauses or terms is feasible in non-deterministic polynomial time, since we have already shown this to hold for **D1** (Lemma 3.1.4).

To show NP-hardness, we simply remark that the $\mathcal{K}_n$ formula $\varphi_{\mathcal{U},\mathcal{S}}$ which was used in the proof of Lemma 3.1.4 is a term with respect to **D2**. This means we can use exactly the same proof of NP-hardness that we used for **D1** to show NP-hardness of entailment between **D2**-terms. We can also use the same reasoning as in the proof for **D1** to transfer the NP-hardness result from terms to clauses.

For the other properties, refer to preceding discussion. □

Definitions D3a and D3b

Given that even very inexpressive definitions like **D2** fail to gain us polynomial behavior, it seems reasonable to explore some more expressive options which allow us to capture all of the expressivity of $\mathcal{K}_n$. We begin with the following definition of clauses that was proposed in [EF89] for the purpose of modal resolution:

$$
\begin{array}{ll}
\textbf{D3} & C \ ::= \top \,|\, \bot \,|\, a \,|\, \neg a \,|\, \Box_i C \,|\, \Diamond_i ConjC \,|\, C \vee C \\
 & ConjC \ ::= C \,|\, ConjC \wedge ConjC
\end{array}
$$

This definition of clauses can be extended to a definition of terms and literals which satisfies **P3** or **P4**, but there is no extension which satisfies both properties, as the following theorem demonstrates:

Theorem 3.1.8.
*There is no definition of literals, clauses, and terms which satisfies both **P3** and **P4** and agrees with **D3** on the set of clauses.*

Proof. Let us suppose for a contradiction that we have a definition of literals, clauses, and terms which satisfies **P3** and **P4** and defines the same set of clauses as **D3**. Then it must be the case that the set of literals is defined as follows:

$$L ::= \top \mid \bot \mid a \mid \neg a \mid \Box_i C \mid \Diamond_i ConjC$$

Now consider the literal $\Diamond_1(a \vee b)$. Because of property **P4**, there must be some literal θ which is equivalent to $\neg(\Diamond_1(a \vee b)) \equiv \Box_i(\neg a \wedge \neg b)$. Clearly, θ must be of the form $\Box_1 \theta'$ for some clause θ', since neither propositional literals nor $\Diamond$-formulae can be equivalent to $\Box_i(\neg a \wedge \neg b)$ (by Theorem 2.3.3). This means that the clause θ' must be equivalent to $\neg a \wedge \neg b$. By Theorem 2.3.2, θ' can only contain propositional disjuncts and unsatisfiable $\Diamond$-formulae. However, a single propositional literal cannot imply both $\neg a$ and $\neg b$, so θ' must have only unsatisfiable disjuncts, contradicting the fact that $\theta' \equiv \neg a \wedge \neg b$. This means that there is no clause θ' which is equivalent to $\neg a \wedge \neg b$, and hence no literal θ equivalent to $\neg(\Diamond_1(a \vee b))$, contradicting our earlier assumption that **P4** was satisfied. □

Let us now consider one of the possible extensions of **D3** which satisfies **P4** and a maximal subset of **P1**-**P7**:

D3a
$$\begin{aligned} L &::= \top \mid \bot \mid a \mid \neg a \mid \Box_i L \mid \Diamond_i L \\ C &::= \top \mid \bot \mid a \mid \neg a \mid \Box_i C \mid \Diamond_i ConjC \mid C \vee C \\ ConjC &::= C \mid ConjC \wedge ConjC \\ T &::= \top \mid \bot \mid a \mid \neg a \mid \Box_i DisjT \mid \Diamond_i T \mid T \wedge T \\ DisjT &::= T \mid DisjT \vee DisjT \end{aligned}$$

It can be seen from inspection of definition **D3a** that it satisfies **P1**. Property **P4** holds by construction, property **P5** is a consequence of Proposition 1.3 in [EF89], and property **P6** holds since **D3a** defines a context-free grammar. As definition **D3a** satisfies **P1** and **P5**, it follows from Theorem 3.1.1 that property **P2** cannot hold. **D3a** also falsifies **P3** as there are clauses that are not disjunctions of literals – take for instance the clause $\Box(a \vee b)$. Given that definition **D3a** is strictly more expressive than definitions **D1** and **D2**, it follows that entailment between clauses

or terms must be NP-hard, which means that **D3a** does not satisfy **P7**. In fact, we can show that entailment between clauses or terms of definition **D3a** is PSPACE-complete, and hence of the same complexity as entailment between arbitrary $\mathcal{K}_n$ formulae.

(i') q_0
(ii') $\bigwedge_{i=0}^{m}(\bigwedge_{j\neq i}((\neg q_i \vee \neg q_j) \wedge \Box(\neg q_i \vee \neg q_j) \wedge ... \wedge \Box^m(\neg q_i \vee \neg q_j)))$
(iiia') $\bigwedge_{i=0}^{m}((\neg q_i \vee \Diamond q_{i+1}) \wedge \Box(\neg q_i \vee \Diamond q_{i+1}) \wedge ... \wedge \Box^m(\neg q_i \vee \Diamond q_{i+1}))$
(iiib') $\bigwedge_{\{i|Q_i=\forall\}} \Box^i(\neg q_i \vee \Diamond(q_{i+1} \wedge p_{i+1})) \wedge \Box^i(\neg q_i \vee \Diamond(q_{i+1} \wedge \neg p_{i+1}))$
(iv') $\bigwedge_{i=1}^{m-1}(\bigwedge_{j=i}^{m-1}(\Box^j(\neg p_i \vee \Box p_i) \wedge \Box^j(p_i \vee \Box\neg p_i)))$
(v') $\Box^m(\neg q_m \vee \theta_1) \wedge .... \wedge \Box^m(\neg q_m \vee \theta_l)$

Figure 3.1: The formula $f'(\beta)$ is the conjunction of the above formulae, where the formulae θ_i in (v') are propositional clauses such that $\theta \equiv \theta_1 \wedge ... \wedge \theta_l$.

Lemma 3.1.9.
*Entailment between clauses (resp. terms) with respect to definition **D3a** is* PSPACE*-hard.*

Proof. Membership in PSPACE is immediate since entailment between arbitrary formulae in $\mathcal{K}_n$ can be decided in polynomial space (Corollary 2.5.2).

To prove PSPACE-hardness, we adapt the proof of PSPACE-hardness of $\mathcal{K}$ outlined in Chapter 2. Specifically, we show how the formula $f(\beta)$ (Figure 2.2) which was used to encode the QBF validity problem can be rewritten as a conjunction of **D3a**-clauses. Our modified encoding $f'(\beta)$ is given in Figure 3.1. We claim that the following:

(1) $f(\beta)$ and $f'(\beta)$ are logically equivalent
(2) if θ is in CNF, then $f'(\beta)$ is a conjunction of clauses with respect to **D3a**
(3) if θ is in CNF, then $f'(\beta)$ can be generated in polynomial time from $f(\beta)$

To show (1), it suffices to show that (i)≡(i'), (ii)≡(ii'), (iiia)≡(iiia'), (iiib)≡(iiib'), (iv)≡(iv'), and (v)≡(v'). The first equivalence is immediate since (i) and (i') are identical. (ii)≡(ii') follows from the fact that $\Box^k(q_i \rightarrow \bigwedge_{j\neq i} \neg q_j) \equiv \bigwedge_{j\neq i} \Box^k(\neg q_i \vee \neg q_j)$. (iiia)≡(iiia') holds since (iiia') is just (iiia) with $q_i \rightarrow \Diamond q_{i+1}$ replaced with $\neg q_i \vee \Diamond q_{i+1}$. We have (iiib)≡(iiib') since $\Box^i(q_i \rightarrow (\Diamond(q_{i+1} \wedge p_{i+1}) \wedge \Diamond(q_{i+1} \wedge \neg p_{i+1}))) \equiv \Box^i(\neg q_i \vee \Diamond(q_{i+1} \wedge p_{i+1})) \wedge \Box^i(\neg q_i \vee \Diamond(q_{i+1} \wedge \neg p_{i+1}))$. The equivalence (iv)≡(iv') holds as $\Box^j((p_i \rightarrow \Box p_i) \wedge (\neg p_i \rightarrow \Box\neg p_i)) \equiv \Box^j(\neg p_i \vee \Box p_i) \wedge \Box^j(p_i \vee \Box\neg p_i)$. Finally, we have (v)≡(v') since $\theta \equiv \theta_1 \wedge ... \wedge \theta_l$. Thus, $f(\beta)$ and $f'(\beta)$ are logically equivalent.

To prove (2), we show that each of the component formulae in $f'(\beta)$ is a conjunction of clauses with respect to **D3a**, provided that θ is in CNF. Clearly this is the case for (i') as (i') is a propositional literal. The formula (ii') is also a conjunction of clauses with respect to **D3a** since it is a conjunction formulae of the form $\Box^k(\neg q_i \vee \neg q_j)$. Similarly, (iiia'), (iiib'), and (iv') are all conjunctions of clauses since the formulae $\Box^k(\neg q_i \vee \Diamond q_{i+1})$, $\Box^i(\neg q_i \vee \Diamond(q_{i+1} \wedge p_{i+1}))$, $\Box^i(\neg q_i \vee \Diamond(q_{i+1} \wedge \neg p_{i+1}))$, $\Box^k(\neg p_i \vee \Box p_i)$, and $\Box^k(p_i \vee \Box \neg p_i)$ are all clauses with respect to **D3a**. The formula (v') must also be a conjunction of clauses since the θ_i are assumed to be propositional clauses, making each $\Box^m(\neg q_m \vee \theta_i)$ a clause with respect to **D3a**, and (v') a conjunction of clauses with respect to **D3a**.

For (3), it is clear that we can transform (i), (iiia), (iiib), and (iv) into (i'), (iiia'), (iiib'), and (iv') in polynomial time as the transformations involve only simple syntactic operations and the resulting formulae are at most twice as large. The transformation from (ii) to (ii') is very slightly more involved, but it is not too hard to see the resulting formula is at most m times as large as the original (and m can be no greater than the length of $f(\beta)$). The only step which could potentially result in an exponential blow-up is the transformation from (v) to (v'), as we put θ into CNF. But under the assumption that θ is already in CNF, the transformation can be executed in polynomial time and space, as all we have to do is separate θ into its conjuncts and rewrite the $(q_m \rightarrow \theta_i)$ as $(\neg q_m \vee \theta_i)$.

Now let $\beta = Q_1 p_1 ... Q_m p_m \theta$ be a QBF such that $\theta = \theta_1 \wedge ... \wedge \theta_l$ for some propositional clauses θ_i. Let $f'(\beta)$ be the formula as defined in Figure 3.1. By (2) above, we know that $f'(\beta) = \lambda_1 \wedge ... \wedge \lambda_p$ for some clauses λ_i with respect to **D3a**. Now consider the formula $\zeta = \Diamond(\Box\lambda_1 \wedge ... \wedge \Box\lambda_p \wedge \Diamond\top)$. We can show that $f'(\beta)$ is satisfiable if and only if ζ is satisfiable as follows:

$$
\begin{aligned}
& \zeta \text{ is unsatisfiable} \\
\Leftrightarrow\ & \Box\lambda_1 \wedge ... \wedge \Box\lambda_p \wedge \Diamond\top \text{ is unsatisfiable} \\
\Leftrightarrow\ & \lambda_1 \wedge ... \wedge \lambda_p \wedge \top \text{ is unsatisfiable} \\
\Leftrightarrow\ & \lambda_1 \wedge ... \wedge \lambda_p \text{ is unsatisfiable} \\
\Leftrightarrow\ & f'(\beta) \text{ is unsatisfiable}
\end{aligned}
$$

But we also know from (1) above that $f'(\beta) \equiv f(\beta)$, and from the proof of Theorem 2.5.1 that $f(\beta)$ is satisfiable just in the case that β is a QBF validity. It is also easy to see that ζ is satisfiable if and only if ζ does not entail the contradiction $\bot$. Putting this altogether, we find that β is valid just in the case that ζ does not entail $\bot$. As ζ and $\bot$ are both clauses and terms with respect to **D3a**, we have shown that the QBF-validity problem for QBF with propositional formulae in

CNF can be reduced to the problems of entailment of clauses or terms with respect to **D3a**. Moreover, this is a polynomial time reduction since it follows from (3) that the transformation from β to ζ can be accomplished in polynomial time. This suffices to show PSPACE-hardness, since it is well-known that QBF-validity remains PSPACE-hard even when we restrict the propositional part θ to be a formula in CNF (cf. [Pap94]).

As for the complexity of entailment between **D3a**-terms, we simply remark that $\lambda \models \bot$ just in the case that $\top \models \neg\lambda$ (Theorem 2.3.1). As the NNF of the negation of **D3a**-clause is a **D3a**-term (this can be shown by a very simple inductive argument), it follows that we can reduce clausal entailment to entailment between terms, making the later problem PSPACE-hard, and thus PSPACE-complete. □

Remark 3.1.10.
If we decide not to include $\top$ and $\bot$ in $\mathcal{K}$, then we must modify the proof of Lemma 3.1.9 by replacing $\top$ with some tautologous **D3a**-clause (e.g. $\Box(a \vee \neg a)$) and $\bot$ with some unsatisfiable **D3a**-clause (e.g. $\Diamond(a \wedge \neg a)$).

Theorem 3.1.11.
*Definition **D3a** satisfies properties **P1**, and **P4**-**P6**, and it falsifies properties **P2**, **P3**, and **P7**.*

Proof. Follows from Lemma 3.1.9 and the preceding discussion. □

If instead we extend **D3** so as to enforce property **P3**, we obtain the following definition:

D3b

$$
\begin{aligned}
L &::= \top \mid \bot \mid a \mid \neg a \mid \Box_i C \mid \Diamond_i ConjC \\
C &::= L \mid C \vee C \\
ConjC &::= C \mid ConjC \wedge ConjC \\
T &::= L \mid T \wedge T
\end{aligned}
$$

Definition **D3b** satisfies all of the properties except **P2**, **P4**, and **P7**. Property **P4** fails to hold because the negation of the literal $\Diamond_1(a \vee b)$ is not equivalent to any literal. Property **P7** fails to hold for the same reasons as for definition **D3a**. To prove that **P5** holds, we use standard logical equivalences to rewrite formulae as equivalent conjunctions of clauses and disjunctions of terms.

Theorem 3.1.12.
*Definition **D3b** satisfies properties **P1**, **P3**, **P5**, and **P6**, and falsifies **P2**, **P4**, and **P7**.*

Proof. The satisfaction of properties **P1** and **P3** can be immediately determined by inspection of definition **D3b**, as can the dissatisfaction of property **P2**. It was shown in the proof of Theorem 3.1.8 that the negation of the literal $\Diamond_1(a \vee b)$ is not equivalent to any literal, which means property **P4** is falsified. We will prove later in this subsection that property **P5** holds for definition **D5**, and we will transfer the result to **D3b** by showing that clauses and terms with respect to **D5** are also clauses and terms with respect to **D3b**. Property **P6** follows from the tractability of recognition for context-free grammars [You67].

For **P7**, we first remark that **D3a** and **D3b** define exactly same set of clauses, which means that we can use the same proof as was used for **D3a** to show that entailment between **D3b**-clauses is PSPACE-complete. For **D3a**-terms, we use the fact that the formula ζ used in the reduction for **D3a**-clauses is in fact a term with respect to **D3b** (this can be easily verified). Thus, the proof of PSPACE-hardness of entailment between **D3a**-clauses also gives us the PSPACE-hardness of entailment between **D3b**-terms. □

Definition D4

The next definition which we will consider is a very simple definition that satisfies properties **P3**, **P4**, and **P5**. The definition, which is inspired by the notion of modal atom proposed in [GS96], defines literals as the set of formulae in NNF that cannot be decomposed propositionally.

D4

$$
\begin{aligned}
L &::= \top \mid \bot \mid a \mid \neg a \mid \Box_i \varphi \mid \Diamond_i \varphi \\
C &::= L \mid C \vee C \\
T &::= L \mid T \wedge T \\
\varphi &::= a \mid \neg a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \Box\varphi \mid \Diamond\varphi
\end{aligned}
$$

Definition **D4** can be shown to satisfy all of the properties except **P2** and **P7**. For **P7**, we note that an arbitrary formula φ in NNF is unsatisfiable (a PSPACE-complete problem) if and only if $\Diamond\varphi \models \bot$.

Theorem 3.1.13.
*Definition **D4** satisfies properties **P1**, **P3**-**P6**, and it falsifies properties **P2** and **P7**.*

Proof. The satisfaction or dissatisfaction of properties **P1**, **P2**, and **P3** can be verified immediately from inspection of definition **D4**.

The proof of property **P4** is straightforward. First, the proof for literals: the negation of the literal a is equivalent to the literal $\neg a$; the negation of the lit-

eral $\neg a$ is equivalent to the literal a; the negation of the literal $\Box_i \varphi$ is equivalent to $\Diamond_i \mathbf{Nnf}(\neg\varphi)$ (which is a literal since $\mathbf{Nnf}(\neg\varphi)$ is a formula in NNF which is equivalent to $\neg\varphi$); and the negation of the literal $\Diamond_i \varphi$ is equivalent to the literal $\Box_i \mathbf{Nnf}(\neg\varphi)$. Then to show the result for clauses and terms, we simply use the fact that negations of disjunctions of formulae are equivalent to conjunctions of the negations of their disjuncts and the negations of conjunctions of formulae are equivalent to the disjunctions of the negations of their conjuncts.

For **P5**, we note that the procedure **Dnf** from Chapter 2 can be used to rewrite any arbitrary $\mathcal{K}_n$ formula as an equivalent disjunction of terms with respect to **D4**. The procedure **Cnf** can be used to transform $\mathcal{K}_n$ formulae into equivalent conjunctions of **D4**-clauses. Note that both **Dnf** and **Cnf** return formulae with the same signature and depth as the original formula (we will require this fact in a later chapter).

For **P6**, we use the tractability of membership for context-free grammars, and for **P7**, we refer to the reduction outlined in the preceding discussion. □

Definition D5

Definition **D4** is very liberal, imposing almost no structure on the formulae behind modal operators. If we define literals to be the formulae in NNF that cannot be decomposed *modally* (instead of propositionally), we obtain a much more restricted definition which satisfies exactly the same properties as **D4**.

$$\begin{array}{ll} & L ::= \top \mid \bot \mid a \mid \neg a \mid \Box_i C \mid \Diamond_i T \\ \textbf{D5} & C \ ::= L \mid C \vee C \\ & T \ ::= L \mid T \wedge T \end{array}$$

To prove that **P5** holds, we show how arbitrary $\mathcal{K}_n$ formulae can be rewritten as conjunctions of clauses or disjunctions of terms with respect to definition **D5** by using standard logical equivalences.

Lemma 3.1.14.
*Definition **D5** satisfies **P5**.*

Proof. We demonstrate that any formula in $\mathcal{K}_n$ in NNF is equivalent to a formula in conjunction of clauses with respect to definition **D5**. The restriction to formulae in NNF is without loss of generality as every formula is equivalent to a formula in NNF (cf. Theorem 2.4.2). The proof proceeds by induction on the structural complexity of formulae. The base case is propositional literals, which are already conjunctions of clauses since every propositional literal is a clause with respect to

D5. We now suppose that the statement holds for formulae ψ_1 and ψ_2 and show that it holds for more complex formulae.

We first consider $\varphi = \psi_1 \wedge \psi_2$. By assumption, we can find clauses ρ_i and ζ_j such that $\psi_1 \equiv \rho_1 \wedge ... \wedge \rho_n$ and $\psi_2 \equiv \zeta_1 \wedge ... \wedge \zeta_m$. Thus, φ is equivalent to the formula $\rho_1 \wedge ... \wedge \rho_n \wedge \zeta_1 \wedge ... \wedge \zeta_m$, which is a conjunction of clauses with respect to definition **D5**.

Next we consider $\varphi = \psi_1 \vee \psi_2$. By the induction hypothesis, we have $\psi_1 \equiv \rho_1 \wedge ... \wedge \rho_n$ and $\psi_2 \equiv \zeta_1 \wedge ... \wedge \zeta_m$ for some clauses ρ_i and ζ_j. Thus, $\varphi \equiv (\rho_1 \wedge ... \wedge \rho_n) \vee (\zeta_1 \wedge ... \wedge \zeta_m)$, which can be written equivalently as $\varphi \equiv \wedge_{(i,j) \in \{1,...,n\} \times \{1,...,m\}} (\rho_i \vee \zeta_j)$. Since the union of two clauses produces another clause, all of the $\rho_i \vee \zeta_j$ are clauses, completing the proof.

We now consider the case where $\varphi = \Box_k \psi_1$. By assumption, $\psi_1 \equiv \rho_1 \wedge ... \wedge \rho_n$, where each ρ_i is a clause. So $\varphi \equiv \Box_k (\rho_1 \wedge ... \wedge \rho_n)$. But we also know that $\Box_k (\rho_1 \wedge ... \wedge \rho_n) \equiv \Box_k \rho_1 \wedge ... \wedge \Box_k \rho_n$. It follows that φ is equivalent to $\Box_k \rho_1 \wedge ... \wedge \Box_k \rho_n$, which is a conjunction of clauses since the $\Box_k \rho_i$ are all clauses.

Finally, we consider $\varphi = \Diamond_k \psi_1$. Using the induction hypothesis, we have $\varphi \equiv \Diamond_k (\rho_1 \wedge ... \wedge \rho_n)$ for clauses ρ_i. But since the ρ_i are clauses, each ρ_i is a disjunction of literals $l_{i,1} \vee ... \vee l_{i,p_i}$. After distributing $\wedge$ over $\vee$ and $\vee$ over $\Diamond$, we find that φ is equivalent to the formula

$$\bigvee_{(j_1,...,j_n) \in \{1,...,p_1\} \times ... \times \{1,...,p_n\}} \Diamond_k (l_{1,j_1} \wedge l_{2,j_2} \wedge ... \wedge l_{n,j_n})$$

which is a clause with respect to **D5**.

The proof that every formula is equivalent to a disjunction of terms with respect to **D5** proceeds analogously. □

Theorem 3.1.15.
*Definition **D5** satisfies properties **P1**, **P3**-**P6**, and it falsifies **P2** and **P7**.*

Proof. Properties **P1**-**P3** can be verified by inspection of definition **D5**. We omit the proof for **P4** as it is quite similar (but more tedious) than the proof for **D4**. Property **P5** was proven in Lemma 3.1.14, and for property **P6** we again use the fact that the membership problem for context-free grammars is tractable. For property **P7**, we can show that entailment between clauses or terms is PSPACE-complete by using exactly the same proof as was given for **D3a** (see Lemma 3.1.9). This proof is applicable to **D5** since the formula $f'(\beta)$ which was used to encode instances of QBF is also a conjunction of clauses with respect to **D5**. □

Finally, we close this subsection by showing that the sets of clauses and terms with respect to **D5** are properly included in the sets of clauses and terms for def-

inition **D3b**. We require this result in order to transfer our proof of property **P5** to definition **D3b**.

Theorem 3.1.16.
*Every clause (resp. term) with respect to **D5** is a clause (resp. term) with respect to definition **D3b**.*

Proof. We will show by induction on the structural complexity of formulae that:

1. every clause C with respect to **D5** is a clause with respect to definition **D3b**
2. every term T with respect to **D5** is a term with respect to definition **D3b** *and* a conjunction of clauses with respect to **D3b**

We require this stronger formulation of the statement to prove some of the sub-cases.

The base case for our induction is propositional literals, which are both clauses and terms with respect to **D5**. It is easy to see that (1) and (2) are verified since propositional literals are both clauses and terms with respect to definitions **D3b** (and hence they are also conjunctions of clauses with respect to **D3b**).

For the induction step, we will show that the above statements hold for arbitrary clauses or terms w.r.t. **D5** under the assumption that the statements hold for all of their proper sub-clauses and sub-terms.

We begin with clauses. Let C be a **D5**-clause such that all proper sub-clauses and sub-terms of C satisfy (1) and (2). Now since C is a clause with respect to **D5**, it can either be a propositional literal or a formula of the form $C_1 \vee C_2$ for clauses C_1 and C_2, $\Box_k C_1$ for some clause C_1, or $\Diamond_k T_1$ for some term T_1. The case where C is a propositional literal has already been treated in the base case. Let us thus consider the case where $C = C_1 \vee C_2$. By the induction hypothesis both C_1 and C_2 are clauses with respect to definition **D3b** and for definition **D3b** the disjunction of two clauses is a clause, so statement (1) is verified. We next consider the case where $C = \Box_k C_1$ for some clause C_1 with respect to **D5**. Statement (1) follows easily as we know that C_1 must also be a clause with respect to **D3b**, and for definition **D3b** putting a $\Box$ modality before a clause yields another clause. We now suppose that $C = \Diamond_k T_1$ for some term T_1 with respect to **D5**. We know from the induction hypothesis that T_1 is a conjunction of clauses with respect to **D3b** and hence that $\Diamond_k T_1$ is a clause with respect to **D3b**.

We next consider terms. Let T be a **D5**-term such that all proper sub-clauses and sub-terms of T satisfy (1) and (2). Then T must be either a propositional literal or a formula of the form $T_1 \wedge T_2$ for terms T_1 and T_2, $\Box_k C_1$ for some clause C_1, or $\Diamond_k T_1$ for some term T_1. If $T = T_1 \wedge T_2$, then the first part of (2) holds since we know T_1 and T_2 to be terms with respect to **D3b**, and conjunctions of terms are

also terms for definition **D3b**. The second half is also verified since both T_1 and T_2 are assumed to be conjunctions of clauses with respect to **D3b**, which means that T is also a conjunction of clauses with respect to this definition. Next suppose that $T = \Box_k C_1$. Since C_1 is known to be a clause with respect to **D3b**, the formula $\Box_k C_1$ is a literal, and hence both a term and clause with respect to definition **D3b**. Finally, we treat the case where $T = \Diamond_k T_1$. We use the assumption that T_1 is a conjunction of clauses with respect to **D3b**, from which we get that $\Diamond_k T_1$ is a literal, and hence both a term and a clause with respect to **D3b**. □

3.1.3 Summary and discussion

A summary of our analysis of the different definitions with respect to properties **P1**-**P7** is provided in Figure 3.2.

	D1	**D2**	**D3a**	**D3b**	**D4**	**D5**
P1	yes	yes	yes	yes	yes	yes
P2	yes	yes	no	no	no	no
P3	no	yes	no	yes	yes	yes
P4	yes	yes	yes	no	yes	yes
P5	no	no	yes	yes	yes	yes
P6	yes	yes	yes	yes	yes	yes
P7	no (NP-complete)		no (PSPACE-complete)			

Figure 3.2: Properties of candidate definitions of literals, clauses, and terms.

Deciding between different candidate definitions is of course more complicated than counting up the number of properties that the definitions satisfy, the simple reason being that some properties are more important than others. Take for instance property **P5** which requires clauses and terms to be expressive enough to represent all of the formulae in $\mathcal{K}_n$. If we just use the standard propositional definition of clauses and terms (thereby disregarding the modal operators), then we find that it satisfies every property except **P5**, and hence more properties than any of the definitions considered in this section, and yet we would be hard-pressed to find someone who considers the propositional definition an appropriate definition for $\mathcal{K}_n$. This demonstrates that expressiveness is a particularly important property, so important in fact that we should be willing to sacrifice properties **P2** and **P7** to keep it. Among the definitions that satisfy **P5**, we prefer definitions **D4** and **D5** to definitions **D3a** and **D3b**, as the latter definitions have less in common with the propositional definition and present no advantages over **D4** and **D5**.

Of course, when it comes down to it, the choice of a definition must depend on the particular application in mind. There may very well be circumstances in which a less expressive or less elegant definition may prove to be the most suitable. In this thesis, we are using clauses and terms to define prime implicates and prime implicants, so for us the most important criteria for choosing a definition will be the quality of the notions of prime implicates and prime implicants that the definition induces.

3.2 Defining Prime Implicates and Prime Implicants in $\mathcal{K}_n$

In the previous section, we introduced a number of different possible definitions of clauses and terms in $\mathcal{K}_n$. Each of these definitions gives rise to corresponding notions of prime implicates and prime implicants. The objective of the present section will be to evaluate to what extent the notions of prime implicates and prime implicants induced by the various definitions are suitable generalizations of the propositional notions.

3.2.1 Basic definitions

Once a definition of clauses and terms for $\mathcal{K}_n$ has been fixed, prime implicates and prime implicants can be defined in exactly the same manner as in propositional logic:

Definition 3.2.1.
A clause λ is an implicate of a formula φ if and only if $\varphi \models \lambda$. A clause λ is a *prime implicate* of φ if and only if:

1. λ is an implicate of φ
2. If λ' is an implicate of φ such that $\lambda' \models \lambda$, then $\lambda \models \lambda'$

Definition 3.2.2.
A term κ is an implicant of the formula φ if and only if $\kappa \models \varphi$. A term κ is a *prime implicant* of φ if and only if:

1. κ is an implicant of φ
2. If κ' is an implicant of φ such that $\kappa \models \kappa'$, then $\kappa' \models \kappa$

Of course, the quality of the notion of prime implicate (resp. implicant) that we get will be determined by the definition of clause (resp. term) that we have chosen.

3.2.2 Desirable properties

Our evaluation of the different notions of prime implicates and prime implicants in $\mathcal{K}_n$ will be based on the following set of well-known properties of the propositional notions (cf. [Mar00]):

Finiteness The number of prime implicates (resp. prime implicants) of a formula is finite modulo logical equivalence.

Covering Every implicate of a formula is entailed by some prime implicate of the formula. Similarly, every implicant of a formula entails some prime implicant of the formula.

Equivalence A model $\mathfrak{M}$ is a model of φ if and only if $\mathfrak{M}$ is a model of all the prime implicates of φ if and only if $\mathfrak{M}$ is a model of at least one prime implicant of φ [2].

Implicant-Implicate Duality Every prime implicant of a formula is equivalent to the negation of some prime implicate of the negated formula. Conversely, every prime implicate of a formula is equivalent to the negation of a prime implicant of the negated formula.

Distribution If λ is a prime implicate of $\varphi_1 \vee ... \vee \varphi_n$, then there exist prime implicates λ_1, ..., λ_n of φ_1, ..., φ_n such that $\lambda \equiv \lambda_1 \vee ... \vee \lambda_n$. Likewise, if κ is a prime implicant of $\varphi_1 \wedge ... \wedge \varphi_n$, then there exist prime implicants κ_1, ..., κ_n of φ_1, ..., φ_n such that $\kappa \equiv \kappa_1 \wedge ... \wedge \kappa_n$

Finiteness ensures that the prime implicates/implicants of a formula can be finitely represented, which is of course essential if we aim to use prime implicates/implicants in applications. The **Covering** property requires that the prime implicates provide a complete representation of the formula's implicates (and similarly for implicants), a crucial property when one uses these notions for knowledge compilation. **Equivalence** guarantees that no information is lost in replacing a formula by its prime implicates/implicants. Definitions which satisfy **Finiteness** and **Covering** also satisfy **Equivalence**, but the converse does not necessarily hold. **Implicant-Implicate Duality** allows us to transfer results and algorithms for prime implicates to prime implicants, and vice-versa. Finally, **Distribution** relates the prime implicates/implicants of a formula to the prime implicates/implicants of its sub-formulae. This property will play a key role in the prime implicate generation algorithm presented in the next chapter.

2. The property **Equivalence** is more commonly taken to mean that a formula is equivalent to the conjunction of its prime implicates and the disjunction of its prime implicants. We have chosen a model-theoretic formulation in order to allow for the possibility that the set of prime implicates/implicants is infinite.

3.2.3 Analysis of candidate definitions

In this subsection, we evaluate the notions of prime implicates and prime implicants induced by each of the different candidate definitions of clauses and terms using the criteria set forth in the previous subsection. Our results will show that definition **D4** yields a notion of prime implicates and prime implicants which satisfy all of the stated criteria, and moreover, that it is the only candidate definition with this property.

Analysis of definitions D1 and D2

For definitions **D1** and **D2**, we show that **Equivalence** does not hold.

Theorem 3.2.3.
*The notions of prime implicates and prime implicants induced by definitions **D1** and **D2** do not satisfy **Equivalence**.*

Proof. The proof is the same for both definitions. Suppose that **Equivalence** holds. Then for every formula φ, the set Π of prime implicates of φ is equivalent to φ. But this means that the set $\Pi \cup \{\neg\varphi\}$ is inconsistent, and hence by compactness of $\mathcal{K}_n$ (Theorem 2.7.3) that there is some finite subset $S \subseteq \Pi \cup \{\neg\varphi\}$ which is inconsistent. If $\varphi \not\equiv \bot$, then we know that the set S must contain $\neg\varphi$ because the set of prime implicates of φ cannot be inconsistent. But then the conjunction of elements in $S \setminus \{\neg\varphi\}$ is a conjunction of clauses which is equivalent to φ. It follows that every formula φ is equivalent to some conjunction of clauses. As we have shown earlier in the proof of Theorem 3.1.1 that there are formulae which are not equivalent to a conjunction of clauses with respect to **D1** or **D2**, it follows that **Equivalence** cannot hold for these definitions. □

Analysis of definitions D3a, D3b, and D5

For definitions **D3a**, **D3b**, and **D5**, we will show that the clause $\Box(\Diamond^k a) \vee \Diamond(a \wedge b \wedge \Box^k \neg a)$ is a prime implicate of $\Box(a \wedge b)$ for every $k \geq 1$. We thereby demonstrate not only that these definitions admit formulae with infinitely many prime implicates but also that they allow seemingly irrelevant clauses to be counted as prime implicates. This gives us strong grounds for dismissing these definitions as much of the utility of prime implicates in applications comes from their ability to eliminate such irrelevant consequences.

Theorem 3.2.4.
*The notions of prime implicates and prime implicants induced by definitions **D3a**, **D3b**, and **D5** do not satisfy **Finiteness**.*

Proof. Suppose that clauses are defined with respect to definition **D3a**, **D3b**, or **D5** (the proof is the same for all three definitions). Consider the formula $\varphi = \Box(a \wedge b)$. It follows from Theorem 2.3.3 that φ implies $\lambda_k = \Box(\Diamond^k a) \vee \Diamond(a \wedge b \wedge \Box^k \neg a)$ for every $k \geq 1$. As the formulae λ_k are clauses (with respect to **D3a**, **D3b**, and **D5**), the λ_k are all implicates of φ. To complete the proof, we show that every λ_k is a prime implicate of φ. Since the λ_k are mutually non-equivalent (because $\Box^p \neg a \not\models \Box^q \neg a$ whenever $p \neq q$), it follows that φ has infinitely many prime implicates modulo equivalence.

Consider some λ_k and some implicate $\mu = \Diamond\psi_1 \vee ... \vee \Diamond\psi_m \vee \Box\chi_1 \vee ... \vee \Box\chi_n$ of φ that implies it (by Theorem 2.3.2 there cannot be any propositional literals in μ). Using Theorem 2.3.3 and the fact that $\varphi \models \mu \models \lambda_k$, we get the following:

(a) $a \wedge b \models \chi_i \vee \psi_i \vee ... \vee \psi_m$ for some χ_i

(b) $\chi_i \models (\Diamond^k a) \vee (a \wedge b \wedge \Box^k \neg a)$ for every χ_i

(c) $\psi_1 \vee ... \vee \psi_m \models a \wedge b \wedge \Box^k \neg a$

Let χ_i be such that $a \wedge b \models \chi_i \vee \psi_i \vee ... \vee \psi_m$. We remark that χ_i must be satisfiable since otherwise we can combine (a) and (c) to get $a \wedge b \models a \wedge b \wedge \Box^k \neg a$. Now by (b), we know that $\chi_i \models (\Diamond^k a) \vee (a \wedge b \wedge \Box^k \neg a)$ and hence that $\chi_i \wedge (\Box^k \neg a) \wedge (\neg a \vee \neg b \vee \Diamond^k a)$ is inconsistent. It follows that both $\chi_i \wedge (\Box^k \neg a) \wedge \neg a$ and $\chi_i \wedge (\Box^k \neg a) \wedge \neg b$ are inconsistent. Using Theorem 2.3.1, we find that either $\chi_i \models \Diamond^k a$ or $\chi_i \models a \wedge b$. As χ_i is a satisfiable clause with respect to definitions **D3a**, **D3b**, and **D5**, it cannot imply $a \wedge b$, so we must have $\chi_i \models \Diamond^k a$. By putting (a) and (c) together, we find that

$$a \wedge b \wedge \neg\chi_i \models \psi_1 \vee ... \vee \psi_m \models a \wedge b \wedge \Box^k \neg a$$

It follows that $\neg\chi_i \models \Box^k \neg a$, i.e. $\Diamond^k a \models \chi_i$. We thus have $\chi_i \equiv \Diamond^k a$ and $\psi_1 \vee ... \vee \psi_m \equiv a \wedge b \wedge \Box^k \neg a$. As $\Diamond^k a \models \chi_i$ and $a \wedge b \wedge \Box^k \neg a \models \psi_1 \vee ... \vee \psi_m$, by Theorem 2.3.3 we get $\Box(\Diamond^k a) \vee \Diamond(a \wedge b \wedge \Box^k \neg a) \models \Box\chi_i \vee \Diamond\psi_i \vee ... \vee \Diamond\psi_m \models \mu$ and hence $\lambda_k \equiv \mu$. We have thus shown that any implicate of φ which implies λ_k must be equivalent to λ_k. This means that each λ_k is a prime implicate of φ, completing the proof. □

Analysis of definition D4

We will now show that the notions of prime implicates and prime implicants induced by definition **D4** satisfy all of the desired properties. We start off by proving that **Implicant-Implicate Duality** holds, as we will make use of this result in the proofs of some of the other properties.

Theorem 3.2.5.
*The notions of prime implicates and prime implicants induced by definition **D4** satisfies **Implicant-Implicate Duality**.*

Proof. Suppose for a contradiction that we have a prime implicant κ of some formula φ which is not equivalent to the negation of a prime implicate of $\neg\varphi$. Let λ be a clause which is equivalent to $\neg\kappa$ (there must exist such a clause because of property **P4**, cf. Theorem 3.1.13). The clause λ is an implicate of $\neg\varphi$ since $\kappa \models \varphi$ and $\lambda \equiv \neg\kappa$. Since we have assumed that λ is not a prime implicate, there must be some implicate λ' of $\neg\varphi$ such that $\lambda' \models \lambda$ and $\lambda \not\models \lambda'$. But then let κ' be a term equivalent to $\neg\lambda'$ (here again we use **P4**). Now κ' must be an implicant of φ since $\neg\varphi \models \neg\kappa'$. Moreover, κ' is strictly weaker than κ since $\lambda' \models \lambda$ and $\lambda \not\models \lambda'$ and $\kappa \equiv \neg\lambda$ and $\kappa' \equiv \neg\lambda'$. But this means that κ cannot be a prime implicant, contradicting our earlier assumption. Hence, we can conclude that every prime implicant of a formula φ is equivalent to the negation of some prime implicate of $\neg\varphi$. The proof that every prime implicate of a formula φ is equivalent to the negation of a prime implicant of $\neg\varphi$ proceeds analogously. □

For the proofs of **Finiteness** and **Covering**, we will require the following lemma which allows us to restrict our attention to those implicates/implicants of a formula whose depths are no greater than that of the formula and whose signatures are contained in the signature of the formula.

Lemma 3.2.6.
*Every implicate λ (w.r.t. definition **D4**) of a formula φ is entailed by some implicate λ' (w.r.t. definition **D4**) of φ with $sig(\lambda') \subseteq sig(\varphi)$ and with depth at most $\delta(\varphi)$. Likewise every implicant κ (w.r.t. definition **D4**) of φ entails an implicant κ' (w.r.t. definition **D4**) of φ with $sig(\kappa') \subseteq sig(\varphi)$ and depth at most $\delta(\varphi)$.*

Proof. We intend to show that the following statement holds: for any formula φ and any implicate λ of φ, there exists a clause λ' such that $\varphi \models \lambda' \models \lambda$ and $sig(\lambda') \subseteq sig(\varphi)$ and $\delta(\lambda) \leq \delta(\varphi)$. So let φ be an arbitrary formula, and let λ be some implicate of φ. If φ is a tautology, then we can set $\lambda' = \top$. If $\lambda \equiv \bot$, then we can set $\lambda' = \bot$, as this clause verifies all of the necessary conditions. Now we consider the case where neither φ nor λ is a tautology or a falsehood, and we show how to construct the clause λ'. The first thing we do is use the transformation **Dnf** from Chapter 2 to rewrite φ as a disjunction of terms T_i with respect to **D4** such that the T_i contain only the variables appearing in φ and have depth at most $\delta(\varphi)$:

$$\varphi \equiv T_1 \vee ... \vee T_z$$

As $\varphi \models \lambda$, it must be the case that $T_s \models \lambda$ for every T_s ($1 \leq s \leq z$). Our aim is to find a clause λ_s for each of the terms T_s such that $T_s \models \lambda_s \models \lambda$ and $sig(\lambda_s) \subseteq sig(T_s)$ and $\delta(\lambda_s) \leq \delta(T_s)$. So consider some T_s. Since T_s is a term, it has the form

$$\gamma_1 \wedge ... \wedge \gamma_k \wedge \bigwedge_{i=1}^{n} (\Diamond_i \psi_{i,1} \wedge ... \wedge \Diamond_i \psi_{i,l_i} \wedge \Box_i \chi_{i,1} \wedge ... \wedge \Box_i \chi_{i,m_i})$$

where γ_1, ..., γ_k are propositional literals. As λ is a clause, it must be of the form

$$\rho_1 \vee ... \vee \rho_p \vee \bigvee_{i=1}^{n} (\Diamond_i \epsilon_{i,1} \vee ... \vee \Diamond_i \epsilon_{i,q_i} \vee \Box_i \zeta_{i,1} \vee ... \vee \Box_i \zeta_{i,r_i})$$

where ρ_1, ..., ρ_p are propositional literals. As $T_s \models \lambda$, it must be the case that the formula

$$\begin{array}{c} \gamma_1 \wedge ... \wedge \gamma_k \wedge \bigwedge_{i=1}^{n} (\Diamond_i \psi_{i,1} \wedge ... \wedge \Diamond_i \psi_{i,l_i} \wedge \Box_i \chi_{i,1} \wedge ... \wedge \Box_i \chi_{i,m_i}) \wedge \\ \neg\rho_1 \wedge ... \wedge \neg\rho_p \wedge \bigwedge_{i=1}^{n} (\Box_i \neg\epsilon_{i,1} \wedge ... \wedge \Box_i \neg\epsilon_{i,q_i} \wedge \Diamond_i \neg\zeta_{i,1} \wedge ... \wedge \Diamond_i \neg\zeta_{i,r_i}) \end{array}$$

is unsatisfiable. It follows from Theorem 2.3.1 that one of the following must hold:

(a) $\gamma_1 \wedge ... \wedge \gamma_k \wedge \neg\rho_1 \wedge ... \wedge \neg\rho_p \models \bot$

(b) there exists some $1 \leq i \leq n$ and some $1 \leq u \leq l_i$ such that $\psi_{i,u} \wedge \chi_{i,1} \wedge ... \wedge \chi_{i,m_i} \wedge \neg\epsilon_{i,1} \wedge ... \wedge \neg\epsilon_{i,q_i} \models \bot$

(c) there exists some $1 \leq i \leq n$ and some $1 \leq u \leq r_i$ such that $\neg\zeta_{i,u} \wedge \chi_{i,1} \wedge ... \wedge \chi_{i,m_i} \wedge \neg\epsilon_{i,1} \wedge ... \wedge \neg\epsilon_{i,q_i} \models \bot$

Now if (a) holds, then there must be u and v such that $\gamma_u = \rho_v$. We can then set $\lambda_s = \gamma_u$ since $T_s \models \gamma_u \models \lambda$, $\delta(\gamma_u) = 0 \leq \delta(T_s)$, and $sig(\gamma_u) = \{\gamma_u\} \subseteq sig(T_s)$. If it is (b) that holds, then it must be the case that

$$\psi_{i,u} \wedge \chi_{i,1} \wedge ... \wedge \chi_{i,m_i} \models \epsilon_{i,1} \vee ... \vee \epsilon_{i,q_i}$$

and hence that

$$\Diamond_i (\psi_{i,u} \wedge \chi_{i,1} \wedge ... \wedge \chi_{i,m_i}) \models \Diamond_i \epsilon_{i,1} \vee ... \vee \Diamond_i \epsilon_{i,q_i} \models \lambda$$

We can set $\lambda_s = \Diamond_i (\psi_{i,u} \wedge \chi_{i,1} \wedge ... \wedge \chi_{i,m_i})$, since $T_s \models \Diamond_i (\psi_{i,u} \wedge \chi_{i,1} \wedge ... \wedge \chi_{i,m_i}) \models \lambda$, $\delta(\Diamond_i (\psi_{i,u} \wedge \chi_{i,1} \wedge ... \wedge \chi_{i,m_i})) \leq \delta(T_s)$, and $sig(\Diamond_i (\psi_{i,u} \wedge \chi_{i,1} \wedge ... \wedge \chi_{i,m_i})) \subseteq sig(T_s)$. Finally, if (c) holds, then it must be the case that

$$\chi_{i,1} \wedge ... \wedge \chi_{i,m_i} \models \epsilon_{i,1} \vee ... \vee \epsilon_{i,q_i} \vee \zeta_{i,u}$$

and hence that

$$\Box(\chi_{i,1} \wedge ... \wedge \chi_{i,m_i}) \models \Diamond_i \epsilon_{i,1} \vee ... \vee \Diamond_i \epsilon_{i,q_i} \vee \Box_i \zeta_{i,u} \models \lambda$$

So we can set $\lambda_s = \Box(\chi_{i,1} \wedge ... \wedge \chi_{i,m_i})$, as $T_s \models \Box(\chi_{i,1} \wedge ... \wedge \chi_{i,m_i}) \models \lambda$, $\delta(\Box(\chi_{i,1} \wedge ... \wedge \chi_{i,m_i})) \leq \delta(T_s)$, and $sig(\Box(\chi_{i,1} \wedge ... \wedge \chi_{i,m_i})) \subseteq sig(T_s)$. Thus, we have shown that for every T_s, there is some λ_s such that $T_s \models \lambda_s \models \lambda$ and $sig(\lambda_s) \subseteq sig(T_s)$ and $\delta(\lambda_s) \leq \delta(T_s)$. But then $\lambda_1 \vee ... \vee \lambda_z$ is a clause implied by every T_s, and hence by φ, and such that $sig(\lambda_s) \subseteq \cup_{s=1}^{z} sig(T_s) \subseteq sig(\varphi)$ and $\delta(\lambda_s) \leq max_s\, \delta(T_s) \leq \delta(\varphi)$.

Now let κ be an implicant of φ, and let λ be the formula $\mathbf{Nnf}(\neg\kappa)$. We know from Theorem 2.4.2 that $\lambda \equiv \neg\kappa$, and it is straightforward to show that λ must be a clause with respect to **D4**. But then λ is an implicate of $\neg\varphi$, so there must be some clause λ' with $sig(\lambda') \subseteq sig(\neg\varphi) = sig(\varphi)$ and depth at most $\delta(\neg\varphi) = \delta(\varphi)$ such that $\neg\varphi \models \lambda' \models \lambda$. Let κ' be $\mathbf{Nnf}(\neg\lambda')$. It can be easily verified that κ' is a term. Moreover, by Theorem 2.4.2, we have $\kappa' \equiv \neg\lambda'$, $sig(\kappa') = sig(\neg\lambda') = sig(\lambda')$, and $\delta(\kappa') = \delta(\neg\lambda') = \delta(\lambda')$. But then κ' is a term such that $sig(\kappa') \subseteq sig(\varphi)$, $\delta(\kappa') \leq \delta(\varphi)$, and $\kappa \models \kappa' \models \varphi$. □

Theorem 3.2.7.
*The notions of prime implicates and prime implicants induced by definition **D4** satisfy **Finiteness**.*

Proof. Consider an arbitrary formula φ. From Lemma 3.2.6, we know that for each prime implicate λ of φ, there must be an implicate λ' of φ containing only those propositional atoms and modal operators appearing in φ and such that $\delta(\lambda') \leq \delta(\varphi)$ and $\lambda' \models \lambda$. But since λ is a prime implicate, we must also have $\lambda \models \lambda'$ and hence $\lambda \equiv \lambda'$. Thus, every prime implicate of φ is equivalent to some clause built from the finite set of propositional symbols and modal operators appearing in φ and having depth at most $\delta(\varphi)$. As there are only finitely many non-equivalent formulae on a finite alphabet and with fixed depth, it follows that there can be only finitely many distinct prime implicates. By Theorem 3.2.5, every prime implicant of φ is equivalent to the negation of some prime implicate of $\neg\varphi$. It follows then that every formula can only have finitely many distinct prime implicants. □

Theorem 3.2.8.
*The notions of prime implicates and prime implicants induced by definition **D4** satisfy **Covering**.*

Proof. Let φ be an arbitrary formula. From Lemma 3.2.6, we know that every implicate of φ is entailed by some implicate of φ whose signature is contained in $sig(\varphi)$ and whose depth is at most $\delta(\varphi)$. Now consider the following set

$$\Sigma = \{\sigma \,|\, \varphi \models \sigma, \sigma \text{ is a clause}, sig(\sigma) \subseteq sig(\varphi), \delta(\sigma) \leq \delta(\varphi)\}$$

and define another set Π from Σ as follows:

$$\Pi = \{\sigma \in \Sigma \mid \nexists \sigma' \in \Sigma.\ \sigma' \models \sigma \text{ and } \sigma \not\models \sigma'\}$$

In other words, Π is the set of all of the logically strongest implicates of φ having depth at most $\delta(\varphi)$ and built from the propositional letters and modal operators in φ. We claim the following:

(1) every $\pi \in \Pi$ is a prime implicate of φ

(2) for every implicate λ of φ, there is some $\pi \in \Pi$ such that $\pi \models \lambda$

We begin by proving (1). Suppose that (1) does not hold, that is, that there is some $\pi \in \Pi$ which is not a prime implicate of φ. Since π is by definition an implicate of φ, it follows that there must be some implicate λ of φ such that $\lambda \models \pi$ and $\pi \not\models \lambda$. But by Lemma 3.2.6, there is some implicate λ' of φ such that $\delta(\lambda') \leq \delta(\varphi)$, $sig(\lambda') \subseteq sig(\varphi)$, and $\lambda' \models \lambda$. But that means that λ' is an element of Σ which implies but is not implied by π, contradicting the assumption that π is in Π. We can thus conclude that every element of Π must be a prime implicate of φ.

For (2): let λ be some implicate of φ. Then by Lemma 3.2.6, there exists some clause $\lambda' \in \Sigma$ such that $\lambda' \models \lambda$. If $\lambda' \in \Pi$, we are done. Otherwise, there must exist some $\sigma \in \Sigma$ such that $\sigma \models \lambda'$ and $\lambda' \not\models \sigma$. If $\sigma \in \Pi$, we are done, otherwise, we find another stronger member of Σ. But as Σ has finitely many elements modulo equivalence, after a finite number of steps, we will find some element which is in Π and which implies λ. Since we have just seen that all members of Π are prime implicates of φ, it follows that every implicate of φ is implied by some prime implicate of φ.

For the second part of **Covering**, let κ be an implicant of φ, and let λ be a clause equivalent to $\neg\kappa$ (there must be one because **D4** satisfies property **P4**). Now since $\kappa \models \varphi$, we must also have $\neg\varphi \models \lambda$. According to what we have just shown, there must be some prime implicate π of $\neg\varphi$ such that $\neg\varphi \models \pi \models \lambda$. By Theorem 3.2.5, π must be equivalent to the negation of some prime implicant ρ of φ. But since $\rho \equiv \neg\pi$ and $\pi \models \lambda$ and $\lambda \equiv \neg\kappa$, it follows that $\kappa \models \rho$, completing the proof.□

We now prove that **Equivalence** is satisfied.

Theorem 3.2.9.
*The notions of prime implicates and prime implicants induced by **D4** satisfy **Equivalence**.*

Proof. Let φ be some formula in $\mathcal{K}$, and suppose that $\mathfrak{M}$ is a model of every prime implicate of φ. As **D4** is known to satisfy property **P5** (by Theorem 3.1.13), we can find a conjunction of clauses which is equivalent to φ. By **Covering** (Theorem

3.2.8), each of these clauses is implied by some prime implicate of φ, so $\mathfrak{M}$ must be a model of each of these clauses. It follows that $\mathfrak{M}$ is a model of φ. For the other direction, we simply note that by the definition of prime implicates if $\mathfrak{M}$ is a model of φ, then it must also be a model of every prime implicate of φ. We have thus shown that $\mathfrak{M}$ is a model of φ if and only if it is a model of every prime implicate of φ. Using a similar argument, we can show that $\mathfrak{M}$ is a model of φ if and only if it is a model of some prime implicant of φ. □

Finally, we show that **Distribution** holds.

Theorem 3.2.10.
*The notions of prime implicates and prime implicants induced by definition **D4** satisfy **Distribution**.*

Proof. Let λ be a prime implicate of $\varphi_1 \vee ... \vee \varphi_m$. Now for each φ_i, we must have $\varphi_i \models \lambda$. From **Covering** (Theorem 3.2.8), we know that there must exist some prime implicate λ_i for each φ_i such that $\lambda_i \models \lambda$. This means that the formula $\lambda_1 \vee ... \vee \lambda_m$ (which is a clause because it is a disjunction of clauses) entails λ. But since λ is a prime implicate, it must also be the case that $\lambda \models \lambda_1 \vee ... \vee \lambda_m$, and hence $\lambda \equiv \lambda_1 \vee ... \vee \lambda_m$. The proof for prime implicants is entirely similar. □

We close this subsubsection with some examples which illustrate the notions of prime implicates and prime implicants that one obtains from definition **D4**.

Example 3.2.11.
Consider the following formula φ

$$(a \vee b) \wedge \Box_1(\neg b \vee c) \wedge (b \vee \Diamond_1 b) \wedge \Diamond_2 a \wedge \Diamond_2 e \wedge \Box_2(b \wedge (a \vee c)) \wedge \Box_2 d$$

The prime implicates of φ w.r.t. definition **D4** are:

$$a \vee b,\ \Box_1(\neg b \vee c),\ b \vee \Diamond_1(b \wedge c),\ \Diamond_2(a \wedge b \wedge d),\ \Diamond_2((a \vee c) \wedge b \wedge d \wedge e),\text{ and } \Box_2((a \vee c) \wedge b \wedge d)$$

plus all clauses equivalent to one of the clauses in this list.

The next example demonstrates how we can leverage the **Distribution** property to help compute prime implicates:

Example 3.2.12.
Let us now consider the NNF of the negation of the formula φ from the previous example:

$$(\neg a \wedge \neg b) \vee \Diamond_1(b \wedge \neg c) \vee (\neg b \wedge \Box_1 \neg b) \vee \Box_2 \neg a \vee \Box_2 \neg e \\ \vee \Diamond_2(\neg b \vee (\neg a \wedge \neg c)) \vee \Diamond_2 \neg d$$

We know from the **Distribution** property that the prime implicates of this disjunction w.r.t. definition **D4** are just the logically strongest disjunctions of prime implicates of the disjuncts. Thus, in order to calculate this formula's prime implicates, we simply need to compute the prime implicates of each of the disjuncts, form the different possible disjunctions, and eliminate weaker elements. All but two of the disjuncts are literals (and hence their own prime implicates), so we only need to compute the prime implicates of the first disjunct (which are $\neg a$ and $\neg b$) and the third disjunct (yielding $\neg b$ and $\Box_1 \neg b$). Then we construct the four possible disjunctions of prime implicates, which are:

- $\neg a \vee \Diamond_1(b \wedge \neg c) \vee \neg b \vee \Box_2 \neg a \vee \Box_2 \neg e \vee \Diamond_2(\neg b \vee (\neg a \wedge \neg c)) \vee \Diamond_2 \neg d$
- $\neg b \vee \Diamond_1(b \wedge \neg c) \vee \neg b \vee \Box_2 \neg a \vee \Box_2 \neg e \vee \Diamond_2(\neg b \vee (\neg a \wedge \neg c)) \vee \Diamond_2 \neg d$
- $\neg a \vee \Diamond_1(b \wedge \neg c) \vee \Box_1 \neg b \vee \Box_2 \neg a \vee \Box_2 \neg e \vee \Diamond_2(\neg b \vee (\neg a \wedge \neg c)) \vee \Diamond_2 \neg d$
- $\neg b \vee \Diamond_1(b \wedge \neg c) \vee \Box_1 \neg b \vee \Box_2 \neg a \vee \Box_2 \neg e \vee \Diamond_2(\neg b \vee (\neg a \wedge \neg c)) \vee \Diamond_2 \neg d$

The first clause is logically weaker than the second, so we eliminate it. The other clauses are all mutually non-implying, so they are all prime implicates of $\neg\varphi$. Moreover, every prime implicate of $\neg\varphi$ is equivalent to one of these three clauses.

Our final example shows how the **Implicate-Implicant Duality** can be used to generate prime implicants from prime implicates:

Example 3.2.13.
Let φ be as defined in Example 3.2.11. By the **Implicate-Implicant Duality**, every prime implicant of φ is equivalent to the negation of some prime implicate of $\neg\varphi$. As we have already computed the prime implicates of $\neg\varphi$ in the previous example, we just need to use the NNF transformation to rewrite the negations of the prime implicates as terms. We obtain the following three terms:

- $b \wedge \Box_1(\neg b \vee c) \wedge b \wedge \Diamond_2 a \wedge \Diamond_2 e \wedge \Box_2(b \wedge (a \vee c)) \wedge \Box_2 d$
- $\neg a \wedge \Box_1(\neg b \vee c) \wedge \Diamond_1 b \wedge \Diamond_2 a \wedge \Diamond_2 e \wedge \Box_2(b \wedge (a \vee c)) \wedge \Box_2 d$
- $b \wedge \Box_1(\neg b \vee c) \wedge \Diamond_1 b \wedge \Diamond_2 a \wedge \Diamond_2 e \wedge \Box_2(b \wedge (a \vee c)) \wedge \Box_2 d$

Conclusion

While the comparison in the first part of the chapter suggested that definition **D5** was at least as suitable as **D4** as a definition of clauses and terms, the results obtained in the second part of the chapter rule out **D5** as a suitable definition for prime implicates and prime implicants. In the following chapters, we will mainly concentrate our attention on the notions of prime implicates and prime implicants

induced by definition **D4**, as these have been shown to be the most satisfactory generalizations of the propositional case. From this point on, we will take the words "clause", "term", and "prime implicate" to mean clause, term, and prime implicate with respect to definition **D4**, except where explicitly stated otherwise.

4

Generating and Recognizing Prime Implicates

Now that we have selected an appropriate definition of prime implicates and prime implicants for $\mathcal{K}_n$, it is time to investigate the computational properties of these notions. The first half of the chapter will be devoted to the study of prime implicate generation, which is the main search problem related to prime implicates. In the second part of the chapter, we will turn our attention to the main decision problem which is that of determining whether a given clause is a prime implicate of a formula.

4.1 Prime Implicate Generation

In this section, we investigate the problem of generating the set of prime implicates of a given formula. This task is important if we want to compile a formula into its set of prime implicates. It is also useful when we want to produce abductive explanations, since by **Implicant-Implicate Duality** (Theorem 3.2.5) any algorithm for generating prime implicates can be straightforwardly adapted into an algorithm for generating prime implicants.

4.1.1 Prime implicate generation in propositional logic

The development of methods for generating prime implicates in propositional logic has been an area of active research, and there exist nowadays quite a large number of different generation algorithms [1]. For the most part, these algorithms

1. See [Mar00] for an excellent survey.

can be classified into one of two approaches:

Resolution-based approach The first procedure for generating prime implicates was introduced by Quine in [Qui55]. Quine's algorithm transforms the input formula into a set of clauses, and then iteratively performs resolution on pairs of clauses until a fixpoint is reached, removing at each step any subsumed clauses. Many improvements to this basic algorithm can be found in the literature (cf. [Tis67], [KT90], [Jac92], [dK92], [del99], and [SdV01]). It should be noted that resolution-based approaches to consequence finding also exist for first-order logic (cf. e.g. [Ino92]).

Distribution-based approach Algorithms in this approach exploit in one manner or another the property **Distribution** which characterizes the prime implicates of a disjunction of formulae in terms of the prime implicates of the disjuncts. Examples of algorithms of this type can be found in [SCL69], [Soc91], [Nga93], [Cas96], [RBM97], and [SP99]. Most of these algorithms involve a transformation of input formula into disjunctive normal form, but some methods (like the one in [RBM97]) can handle arbitrary formulae in negation normal form. In contrast to the resolution-based approach, the distribution-based approach cannot be lifted to first-order logic, which does not satisfy **Distribution**.

The prime implicate generation algorithm we will propose in the next subsection follows the distribution-based approach.

4.1.2 The algorithm GenPI

In Figure 4.1, we present the algorithm **GenPI** which computes the set of prime implicates of a given $\mathcal{K}_n$ formula. The algorithm makes use of the procedure **Dnf**(φ) which was introduced in Chapter 2.

The algorithm **GenPI** works as follows: in Step 1, we check whether φ is unsatisfiable, outputting a contradictory clause if this is the case. For satisfiable φ, we set $\mathcal{T}$ equal to a set of terms whose disjunction is equivalent to φ. Because of **Distribution**, we know that every prime implicate of φ is equivalent to some disjunction of prime implicates of the terms in $\mathcal{T}$. In Step 2, for each satisfiable term T, we set $\Delta(T)$ equal to the propositional literal conjuncts of T ($Prop(T)$) plus the strongest $\Box$-formulae implied by T ($B(T)$) plus the strongest $\Diamond$-formulae implied by T ($D(T)$). It is not too hard to see that every prime implicate of T must be equivalent to one of the elements in $\Delta(T)$. This means that in Step 3 we are guaranteed that every prime implicate of the input formula is equivalent to some candidate prime implicate in CANDIDATES. During the comparison phase in

Algorithm 4.1 GenPI

Input: a formula φ

Output: a set of clauses

(1) If **Sat**(φ)=**no**, return $\bot$. Otherwise, set $\mathcal{T} = \mathbf{Dnf}(\varphi)$.

(2) For each $T \in \mathcal{T}$ such that **Sat**(T)=**yes**:

Initialize $B(T)$ and $D(T)$ to $\emptyset$

For each $1 \leq i \leq n$:

If $Box_i(T) \neq \emptyset$, then

$B(T) = B(T) \cup \{ \Box_i \bigwedge_{\psi \in Box_i(T)} \psi \}$

$D(T) = D(T) \cup \{ \Diamond_i (\gamma \wedge \bigwedge_{\psi \in Box_i(T)} \psi) \mid \gamma \in Diam_i(T) \}$

Else if $Diam_i(T) \neq \emptyset$

$D(T) = D(T) \cup \{ \Diamond_i \gamma \mid \gamma \in Diam_i(T) \}$

Set $\Delta(T) = Prop(T) \cup B(T) \cup D(T)$.

(3) Set Candidates = $\{\bigvee_{T \in \mathcal{T}} \theta_T \mid \theta_T \in \Delta(T)\}$.

(4) For each $\lambda_j \in$ Candidates: remove the clause λ_j from Candidates if **Entails**(λ_k, λ_j)=**yes** for some $k < j$, or if both **Entails**(λ_k, λ_j)=**yes** and **Entails**(λ_j, λ_k)=**no** for some $k > j$.

(5) Return Candidates.

Step 4, non-prime candidates are eliminated, and exactly one prime implicate of each equivalence class will be retained.

Example 4.1.1.

We run the algorithm **GenPI** on the formula $\varphi = a \wedge ((\Diamond_1 (b \wedge c) \wedge \Diamond_1 b) \vee (\Diamond_1 b \wedge \Diamond_1 (c \vee d) \wedge \Box_1 e \wedge \Box_1 f)) \wedge \Box_2 \bot$.

Step 1: As φ is satisfiable, we call the function **Dnf** on φ, and it returns the two terms $T_1 = a \wedge \Diamond_1 (b \wedge c) \wedge \Diamond_1 b \wedge \Box_2 \bot$ and $T_2 = a \wedge \Diamond_1 b \wedge \Diamond_1 (c \vee d) \wedge \Box_1 e \wedge \Box_1 f \wedge \Box_2 \bot$.

Step 2: We have $Prop(T_1) = \{a\}$, $B(T_1) = \{\Box_2 \bot\}$, and $D(T_1) = \{\Diamond_1 (b \wedge c), \Diamond_1 b\}$, so we get

$$\Delta(T_1) = \{a, \Box_2 \bot, \Diamond_1 (b \wedge c), \Diamond_1 b\}$$

For T_2, we have $Prop(T_2) = \{a\}$, $B(T_2) = \{\Box_1 (e \wedge f), \Box_2 \bot\}$, and $D(T_2) = \{\Diamond_1 (b \wedge e \wedge f), \Diamond_1 ((c \vee d) \wedge e \wedge f)\}$, giving us

$$\Delta(T_2) = \{a, \Box_1 (e \wedge f), \Box_2 \bot, \Diamond_1 (b \wedge e \wedge f), \Diamond_1 ((c \vee d) \wedge e \wedge f)\}$$

Step 3: The set Candidates will contain all of the different possible disjunctions of elements in $\Delta(T_1)$ with elements in $\Delta(T_2)$, of which there are 20: $a \vee a$,

$a \vee \Box_1 (e \wedge f)$, $a \vee \Box_2 \bot$, $a \vee \Diamond_1 (b \wedge e \wedge f)$, $a \vee \Diamond_1 ((c \vee d) \wedge e \wedge f)$, $\Box_2 \bot \vee a$, $\Box_2 \bot \vee \Box_1 (e \wedge f)$, $\Box_2 \bot \vee \Box_2 \bot$, $\Box_2 \bot \vee \Diamond_1 (b \wedge e \wedge f)$, $\Box_2 \bot \vee \Diamond_1 ((c \vee d) \wedge e \wedge f)$, $\Diamond_1 (b \wedge c) \vee a$, $\Diamond_1 (b \wedge c) \vee \Box_1 (e \wedge f)$, $\Diamond_1 (b \wedge c) \vee \Box_2 \bot$, $\Diamond_1 (b \wedge c) \vee \Diamond_1 (b \wedge e \wedge f)$, $\Diamond_1 (b \wedge c) \vee \Diamond_1 ((c \vee d) \wedge e \wedge f)$, $\Diamond_1 b \vee a$, $\Diamond_1 b \vee \Box_1 (e \wedge f)$, $\Diamond_1 b \vee \Box_2 \bot$, $\Diamond_1 b \vee \Diamond_1 (b \wedge e \wedge f)$, and $\Diamond_1 b \vee \Diamond_1 ((c \vee d) \wedge e \wedge f)$.

Step 4: We will remove from CANDIDATES the clauses $a \vee \Box_1 (e \wedge f)$, $a \vee \Box_2 \bot$, $a \vee \Diamond_1 (b \wedge e \wedge f)$, $a \vee \Diamond_1 ((c \vee d) \wedge e \wedge f)$, $\Box_2 \bot \vee a$, $\Diamond_1 (b \wedge c) \vee a$, and $\Diamond_1 b \vee a$ since they are all strictly weaker than $a \vee a$. We will also eliminate the clauses $\Box_2 \bot \vee \Box_1 (e \wedge f)$, $\Box_2 \bot \vee \Diamond_1 (b \wedge e \wedge f)$, $\Box_2 \bot \vee \Diamond_1 ((c \vee d) \wedge e \wedge f)$, $\Diamond_1 (b \wedge c) \vee \Box_2 \bot$, and $\Diamond_1 b \vee \Box_2 \bot$, since they are weaker than $\Box_2 \bot \vee \Box_2 \bot$. Finally we will remove the clauses $\Diamond_1 b \vee \Box_1 (e \wedge f)$, $\Diamond_1 b \vee \Diamond_1 (b \wedge e \wedge f)$, and $\Diamond_1 b \vee \Diamond_1 ((c \vee d) \wedge e \wedge f)$ since these clauses are respectively weaker than the clauses $\Diamond_1 (b \wedge c) \vee \Box_1 (e \wedge f)$, $\Diamond_1 (b \wedge c) \vee \Diamond_1 (b \wedge e \wedge f)$ and $\Diamond_1 (b \wedge c) \vee \Diamond_1 ((c \vee d) \wedge e \wedge f)$.

Step 5: **GenPI** will return the five remaining clauses in CANDIDATES, which are $a \vee a$, $\Box_2 \bot \vee \Box_2 \bot$, $\Diamond_1 (b \wedge c) \vee \Box_1 (e \wedge f)$, $\Diamond_1 (b \wedge c) \vee \Diamond_1 (b \wedge e \wedge f)$, and $\Diamond_1 (b \wedge c) \vee \Diamond_1 ((c \vee d) \wedge e \wedge f)$.

4.1.3 Correctness of GenPI

Our algorithm can be shown to be a sound and complete procedure for generating prime implicates.

Lemma 4.1.2.
The algorithm **GenPI** *always terminates.*

Proof. We know from Corollary 2.4.5 that the algorithm **Dnf** always terminates and returns a finite set of formulae. This means that there are only finitely many terms T to consider. For each T, the set $\Delta(T)$ contains only finitely many elements (this is immediate given the definition of $\Delta(T)$), which means that the set CANDIDATES also has finite cardinality. In the final step, we compare at most once each pair of elements in CANDIDATES. As the comparison always terminates, and there are only finitely many pairs to check, it follows that the algorithm **GenPI** terminates. □

Lemma 4.1.3.
The algorithm **GenPI** *outputs exactly the set of prime implicates of the input formula.*

Proof. We first prove that every prime implicate of a term T is equivalent to some element in $\Delta(T)$. Let

$$T = \gamma_1 \wedge ... \wedge \gamma_k \wedge \bigwedge_{i=1}^{n} (\Diamond_i \psi_{i,1} \wedge ... \wedge \Diamond_i \psi_{i,l_i} \wedge \Box_i \chi_{i,1} \wedge ... \wedge \Box_i \chi_{i,m_i})$$

be some term (where γ_1, ..., γ_k are propositional literals), and let

$$\lambda = \rho_1 \vee ... \vee \rho_p \vee \bigvee_{i \in \mathcal{L}} (\Diamond_i \epsilon_{i,1} \vee ... \vee \Diamond_i \epsilon_{i,q_i} \vee \Box_i \zeta_{i,1} \vee ... \vee \Box_i \zeta_{i,r_i})$$

be one of its prime implicates. As $T_s \models \lambda$, it must be the case that the formula

$$\begin{array}{c} \gamma_1 \wedge ... \wedge \gamma_k \wedge \bigwedge_{i=1}^{n} (\Diamond_i \psi_{i,1} \wedge ... \wedge \Diamond_i \psi_{i,l_i} \wedge \Box_i \chi_{i,1} \wedge ... \wedge \Box_i \chi_{i,m_i}) \wedge \\ \neg\rho_1 \wedge ... \wedge \neg\rho_p \wedge \bigwedge_{i=1}^{n} (\Box_i \neg\epsilon_{i,1} \wedge ... \wedge \Box_i \neg\epsilon_{i,q_i} \wedge \Diamond_i \neg\zeta_{i,1} \wedge ... \wedge \Diamond_i \neg\zeta_{i,r_i}) \end{array}$$

is unsatisfiable. It follows from Theorem 2.3.1 that one of the following must hold:

(a) $\gamma_1 \wedge ... \wedge \gamma_k \wedge \neg\rho_1 \wedge ... \wedge \neg\rho_p \models \bot$

(b) there exists some $1 \leq i \leq n$ and some $1 \leq u \leq l_i$ such that $\psi_{i,u} \wedge \chi_{i,1} \wedge ... \wedge \chi_{i,m_i} \wedge \neg\epsilon_{i,1} \wedge ... \wedge \neg\epsilon_{i,q_i} \models \bot$

(c) there exists some $1 \leq i \leq n$ and some $1 \leq u \leq r_i$ such that $\neg\zeta_{i,u} \wedge \chi_{i,1} \wedge ... \wedge \chi_{i,m_i} \wedge \neg\epsilon_{i,1} \wedge ... \wedge \neg\epsilon_{i,q_i} \models \bot$

If (a) holds, then there must be some u and v such that $\gamma_u = \rho_v$. That means $\gamma_u \models \lambda$, so λ must be equivalent to γ_u or else we would have found a stronger implicate, contradicting our assumption that λ is a prime implicate of T. But then the result holds since γ_u is in $\Delta(T)$. If (b) holds, then the formula $\Diamond_i (\psi_{i,u} \wedge \chi_{i,1} \wedge ... \wedge \chi_{i,m_i})$ is an implicate of T which implies λ, so $\lambda \equiv \Diamond_i (\psi_{i,u} \wedge \chi_{i,1} \wedge ... \wedge \chi_{i,m_i})$. We are done since $\Diamond_i (\psi_{i,u} \wedge \chi_{i,1} \wedge ... \wedge \chi_{i,m_i})$ is a member of $\Delta(T)$. Finally we consider the case where (c) holds. In this case, $\Box_i (\chi_{i,1} \wedge ... \wedge \chi_{i,m_i})$ is an implicate of T which implies λ, and so is equivalent to λ (as λ is a prime implicate). But then we have the desired result since $\Box_i (\chi_{i,1} \wedge ... \wedge \chi_{i,m_i})$ is one of the elements in $\Delta(T)$. Thus we can conclude that every prime implicate of a term T is equivalent to some element in $\Delta(T)$. By Theorem 2.4.4, the elements in $\mathbf{Dnf}(\varphi)$ are terms, and their disjunction is equivalent to φ. As **D4** satisfies **Distribution** (Theorem 3.2.10), it follows that every prime implicate of the input φ is equivalent to some element in CANDIDATES. This means that if an element λ_j in CANDIDATES is not a prime implicate of φ, then there is some prime implicate π of φ that implies but is not implied by λ_j, and hence some $\lambda_k \in$ CANDIDATES such that $\lambda_k \models \lambda_j$ and $\lambda_j \not\models \lambda_k$. Thus, during the comparison phase, this clause will be removed from CANDIDATES. Now suppose that the clause λ is a prime implicate of φ. Then we know that there must be some $\lambda_j \in$ CANDIDATES such that $\lambda_j \equiv \lambda$, and moreover, we can choose

λ_j so that there is no λ_k with $k < j$ such that $\lambda_k \models \lambda_j$. When in the final step we compare λ_j with all the clauses λ_k with $k \neq j$, we will never find that $\lambda_k \models \lambda_j$ for $k < j$, nor can we have $\lambda_k \models \lambda_j \not\models \lambda_k$ for some $k > j$, otherwise λ would not be a prime implicate. It follows then that λ_j remains in the set CANDIDATES which is returned by the algorithm. We have thus shown that the output of **GenPI** with input φ is precisely the set of prime implicates of φ. □

Theorem 4.1.4.
The algorithm **GenPI** *always terminates and outputs exactly the set of prime implicates of the input formula.*

Proof. Follows directly from Lemmas 4.1.2 and 4.1.3. □

Finally, we have the following theorem which relates the depths, signatures, and literal subformulae of the prime implicates produced by **GenPI** to the depth, signature, and literal subformulae of the input formula. We will require this result later on in Chapter 6.

Theorem 4.1.5.
Let φ be a formula in NNF. The signatures of the clauses output by **GenPI** *on input φ are contained in the signature of φ. The depths of the clauses output by* **GenPI** *on input φ are no greater than $\delta(\varphi)$. Every literal subformula which appears behind the modal operators in some clause output by* **GenPI** *is also a literal subformula of φ.*

Proof. Suppose that **GenPI** is run on an input formula φ in NNF. We know from Theorem 2.4.6 that $sig(T) \subseteq sig(\varphi)$ and $\delta(T) \leq \delta(\varphi)$ for every $T \in \mathbf{Dnf}(\varphi)$. Moreover, we know from the definition of $\Delta(T)$ that $sig(\theta) \subseteq sig(T) \subseteq sig(\varphi)$ and $\delta(\theta) \leq \delta(T) \leq \delta(\varphi)$ for every $\theta \in \Delta(T)$. It is also easy to see from the definition that if ψ is such that $\Diamond\psi$ or $\Box\psi$ is a conjunct of some element in $\Delta(T)$, then ψ is a conjunction of subformulae of φ, so each of the literal subformulae appearing in ψ must also appear in φ. As the formulae in CANDIDATES are disjunctions of the elements in the $\Delta(T)$, it follows that the formulae in CANDIDATES have signatures contained in $sig(\varphi)$ and depths at most $\delta(\varphi)$ and that the literal subformulae behind their modal operators also appear in φ. This is enough to prove the result since every clause output by **GenPI** belongs to CANDIDATES. □

4.1.4 Bounds on prime implicate size

By examining the prime implicates produced by **GenPI**, we can put an upper bound on the length of a formula's prime implicates.

Theorem 4.1.6.
The smallest clausal representation of a prime implicate of a formula is at most single-exponential in the length of the formula.

Proof. Let φ be a formula in $\mathcal{K}_n$. Prime implicates of φ generated by **GenPI** can have at most $2^{|\varphi|}$ disjuncts as there are at most 2^{φ} terms in $\mathbf{Dnf}(\varphi)$ by Theorem 2.4.6. Moreover, each disjunct has length at most $2|\varphi|$, since the elements in $\Delta(T)$ are never larger than $|T|$ and the length of T cannot exceed $2|\varphi|$ (by Theorem 2.4.6). This gives us a total of $2|\varphi| * 2^{|\varphi|}$ symbols, to which we must add the at most $2^{|\varphi|} - 1$ disjunction symbols connecting the disjuncts. We thus find that the length of the smallest clausal representation of a prime implicate of a formula φ cannot exceed $2|\varphi| * 2^{|\varphi|} + (2^{|\varphi|} - 1)$, which is clearly single-exponential in $|\varphi|$.

For a tighter bound, suppose that $\mathbf{Nnf}(\varphi)$ has at most l mutually non-equivalent literal subformulae appearing outside the scope of modal operators. Then there are at most 2^l mutually non-equivalent elements in $\mathcal{T} = \mathbf{Dnf}(\varphi)$ by Theorem 2.4.6. It follows that we can choose a subset $\mathcal{T}'$ of $\mathcal{T}$ with cardinality at most 2^l such that every element of $\mathcal{T}$ is equivalent to some element of $\mathcal{T}'$. Now let π be some formula in the output of **GenPI**. We know that π is of the form $\bigvee_{T\in\mathcal{T}} \theta_T$ where $\theta_T \in \Delta(T)$ for all $T \in \mathcal{T}$. Now let $\pi' = \bigvee_{T\in\mathcal{T}'} \theta_T$. Clearly $\pi' \models \pi$. We also have $\pi' \models \pi$, since every $T \in \mathcal{T}$ we can find $T' \in \mathcal{T}'$ such that $T \equiv T'$, which means that $T \models \theta_{T'} \models \pi'$. Thus, we have shown that every prime implicate of φ is equivalent to a clause with length at most $2|\varphi| * 2^l + (2^l - 1)$. □

This upper bound is optimal as we can find formulae with exponentially large prime implicates.

Theorem 4.1.7.
The length of the smallest clausal representation of a prime implicate of a formula can be exponential in the length of the formula.

Proof. Consider the formula

$$\varphi = \bigwedge_{i=1}^{n} (\Box a_{i1} \vee \Box a_{i2})$$

and the clause

$$\lambda = \bigvee_{(i_1,\ldots,i_n)\in\{1,2\}^n} \Box(a_{1i_1} \wedge a_{2i_2} \wedge \ldots \wedge a_{ni_n})$$

where $a_{ij} \neq a_{kl}$ whenever $i \neq k$ or $j \neq l$. It is not difficult to see that φ and λ are equivalent, which means that λ must be a prime implicate of φ. All that remains to be shown is that any clause equivalent to λ must have length at least $|\lambda|$. This

yields the result since λ clearly has size exponential in n, whereas the length of φ is only linear in n.

Let λ' be a shortest clause which is equivalent to λ. As λ' is equivalent to λ, it follows from Theorem 2.3.2 that λ' is a disjunction of $\Box$-literals and of inconsistent $\Diamond$-literals. But since λ' is assumed to be a shortest representation of λ, it cannot contain any inconsistent $\Diamond$-literals or any redundant $\Box$-literals, since we could remove them to find an equivalent shorter clause. So λ' must be of the form $\Box\chi_1 \vee ... \vee \Box\chi_m$, where $\chi_l \not\models \chi_j$ whenever $l \neq j$. Now since $\lambda' \models \lambda$, every disjunct $\Box\chi_p$ must also imply λ. As λ is a disjunction of $\Box$-literals, it follows from Theorem 2.3.3 that every disjunct $\Box\chi_p$ of λ' implies some disjunct $\Box\delta_q$ of λ. But that means that every $\Box\chi_p$ must have length at least $2n + 1$, since each χ_p is a satisfiable formula which implies a conjunction of n distinct propositional variables. We also know that every disjunct $\Box\delta_q$ of λ implies some disjunct $\Box\chi_p$ of λ' since $\lambda \models \lambda'$. We now wish to show that no two disjuncts of λ imply the same disjunct of λ'. Suppose that this is not the case, that is, that there are distinct disjuncts $\Box\delta_1$ and $\Box\delta_2$ of λ and some disjunct $\Box\chi_p$ of λ' such that $\Box\delta_1 \models \Box\chi_p$ and $\Box\delta_2 \models \Box\chi_p$. Now since $\Box\delta_1$ and $\Box\delta_2$ are distinct disjuncts, there must be some i such that $\Box\delta_1 \models a_{i1}$ and $\Box\delta_2 \models a_{i2}$ or $\Box\delta_1 \models a_{i2}$ and $\Box\delta_2 \models a_{i1}$. We know that $\Box\chi_p \models \Box\delta_q$ for some δ_q, and that every δ_q implies either a_{i1} or a_{i2}, so either $\Box\chi_p \models \Box a_{i1}$ or $\Box\chi_p \models \Box a_{i2}$. But we know that the $\Box\delta_q$ each imply either $\Box a_{i1}$ or $\Box a_{i2}$ but not both, so one of $\Box\delta_1$ and $\Box\delta_2$ must not imply $\Box\chi_p$. This contradicts our earlier assumption that $\Box\delta_1 \models \Box\chi_p$ and $\Box\delta_2 \models \Box\chi_p$, so each disjunct of λ must imply a distinct disjunct of λ'. We have thus demonstrated that λ' contains just as many disjuncts as λ. As we have already shown that the disjuncts of λ' are no shorter than the disjuncts of λ, it follows that $|\lambda'| \geq |\lambda|$, and hence $|\lambda'| = |\lambda|$. We conclude that every clause equivalent to λ has length at least $|\lambda|$, completing the proof. □

It is interesting to note that the formula used in the proof of Theorem 4.1.7 has a single modal operator and a depth of 1, which means that we cannot avoid this worst-case spatial complexity by restricting our attention to formulae with few modalities or of shallow depth. Nor can we escape this exponential worst-case spatial complexity by dropping down to one of the less expressive notions of prime implicates examined in the previous chapter, as the following theorem demonstrates.

Theorem 4.1.8.
*If prime implicates are defined using either **D1** or **D2**, then the length of the smallest clausal representation of a prime implicate of a formula can be exponential in the length of the formula.*

For Theorem 4.1.8, we will prove that the following clause

$$\lambda = \bigvee_{(q_1,\ldots,q_n)\in\{\Diamond,\Box\}^n} \Box q_1 \ldots q_n\, c$$

is a prime implicate (for both **D1** and **D2**) of the $\mathcal{K}$-formula

$$\begin{aligned}\varphi \quad = \quad & (\,\Box\Diamond(b_0 \wedge b_1) \,\vee\, \Box\Box(b_0 \wedge b_1)\,) \wedge \bigwedge_{i=2}^{n} (\,\Box^i\Diamond b_i \,\vee\, \Box^i\Box b_i\,) \\ & \wedge \bigwedge_{i=1}^{n-1} \Box^{i+1}\,(\,(b_{i-1} \wedge b_i) \rightarrow \Box b_i\,) \,\wedge\, \Box^{n+1}\,(\,(b_{n-1} \wedge b_n) \rightarrow c\,)\end{aligned}$$

and moreover that there is no shorter way to represent λ. To prove this, we will requiring the following technical lemmas.

Lemma 4.1.9.
*Let $l_1 \vee \ldots \vee l_m$ be a **D1**-clause which implies $q_1 \ldots q_n a$, where $q_i \in \{\Box, \Diamond\}$ and a is a propositional variable. Then either $l_1 \vee \ldots \vee l_m \equiv q_1 \ldots q_n a$ or $l_1 \vee \ldots \vee l_m \equiv q_1 \ldots q_j \bot$ for some $1 \leq j \leq n$ or $l_1 \vee \ldots \vee l_m \equiv \bot$.*

Proof. The proof is by induction on the value of n. When $n = 0$, we have just $l_1 \vee \ldots \vee l_m \models a$. According to Theorem 2.3.2, every disjunct of $l_1 \vee \ldots \vee l_m$ must be either a or some unsatisfiable formula. It follows that $l_1 \vee \ldots \vee l_m \equiv a$ or $l_1 \vee \ldots \vee l_m \equiv \bot$.

Now suppose the result holds whenever $n \leq k$, and suppose that we have $l_1 \vee \ldots \vee l_m \models q_1 \ldots q_{k+1} a$. For every l_i, we must have $l_i \models q_1 \ldots q_{k+1} a$, and hence $\models \neg l_i \vee q_1 \ldots q_{k+1} a$. Using Theorem 2.3.1, we arrive at the following four possibilities:

(a) $\models q_1 \ldots q_{k+1} a$
(b) $l_i \equiv \bot$
(c) $q_1 = \Diamond$ and $l_i \equiv \Diamond l_i'$ and $l_i' \models q_2 \ldots q_{k+1} a$
(d) $q_1 = \Box$ and $l_i \equiv \Box l_i'$ and $l_i' \models q_2 \ldots q_{k+1} a$

We can eliminate case (a) since $\not\models q_1 \ldots q_{k+1} a$ for every string of modalities $q_1 \ldots q_{k+1}$. We remark that if (c) holds, then according to the induction hypothesis, $l_i \equiv \Diamond q_1 \ldots q_{k+1} a$ or $l_i \equiv \Diamond q_1 \ldots q_j \bot$ for $j \leq k+1$ or $l_i \equiv \bot$. Similarly, if (d) holds, then either $l_i \equiv \Box q_1 \ldots q_{k+1} a$ or $l_i \equiv \Box q_1 \ldots q_j \bot$ for $j \leq k+1$ or $l_i \equiv \bot$.

Now if there is some l_i which is equivalent to $q_1 \ldots q_{k+1} a$, then we get $l_1 \vee \ldots \vee l_m \equiv q_1 \ldots q_{k+1} a$. If there is no such l_i, then let p be such that there is some $l_i \equiv q_1 \ldots q_p \bot$ but there is no $l_i \equiv q_1 \ldots q_{p-1} \bot$. In this case, we get $l_1 \vee \ldots \vee l_m \equiv q_1 \ldots q_p \bot$. Finally, if all disjuncts are unsatisfiable, we have $l_1 \vee \ldots \vee l_m \equiv \bot$. □

Lemma 4.1.10.
Fix $(q_1, ..., q_n) \in \{\Box, \Diamond\}^n$, *and let* $T =$

$$\Box q_1(b_0 \wedge b_1) \wedge (\bigwedge_{k=2}^{n} \Box^k q_k b_k) \wedge \bigwedge_{k=1}^{n-1} \Box^{k+1}((b_{k-1} \wedge b_k) \rightarrow \Box b_k) \wedge \Box^{n+1}((b_{n-1} \wedge b_n) \rightarrow c)$$

Then $T \models \Box r_1 ... r_n c$ *if and only if* $r_k = q_k$ *for all* $1 \leq k \leq n$.

Proof. We begin by showing that for all $1 \leq i \leq n-1$ that the formula

$$b_{i-1} \wedge b_i \wedge (\bigwedge_{k=i+1}^{n} \Box^{k-i-1} q_k b_k) \wedge (\bigwedge_{k=i}^{n-1} \Box^{k-i}((b_{k-1} \wedge b_k) \rightarrow \Box b_k)) \wedge \Box^{n-i}((b_{n-1} \wedge b_n) \rightarrow c)$$

entails the formula $r_{i+1} ... r_n c$ just in the case that $q_{i+1} ... q_n = r_{i+1} ... r_n$.

The proof is by induction on i. The base case is $i = n-1$. We have

$$b_{n-2} \wedge b_{n-1} \wedge q_n b_n \wedge ((b_{n-2} \wedge b_{n-1}) \rightarrow \Box b_{n-1}) \wedge \Box((b_{n-1} \wedge b_n) \rightarrow c) \models r_n c \quad (4.1)$$

if and only if

$$b_{n-2} \wedge b_{n-1} \wedge q_n b_n \wedge \Box b_{n-1} \wedge \Box((b_{n-1} \wedge b_n) \rightarrow c) \models r_n c$$

if and only if (Theorem 2.3.1) either

$$q_n = \Diamond \text{ and } r_n = \Box \text{ and } b_{n-1} \wedge ((b_{n-1} \wedge b_n) \rightarrow c) \models c$$

or

$$q_n = r_n \text{ and } b_{n-1} \wedge b_n \wedge ((b_{n-1} \wedge b_n) \rightarrow c) \models c$$

As $b_{n-1} \wedge ((b_{n-1} \wedge b_n) \rightarrow c) \not\models c$, we cannot have the first alternative. It follows then that if Equation (4.1) holds, then the second alternative must hold, in which case we get $q_n = r_n$, as desired. For the other direction, we simply note that $b_{n-1} \wedge b_n \wedge ((b_{n-1} \wedge b_n) \rightarrow c) \models c$ is a valid entailment, which means $q_n = r_n$ implies Equation (4.1).

Next let us suppose that the above statement holds for all $1 < j \leq i \leq n-1$, and let us prove the statement holds when $i = j - 1$. Then

$$b_{j-2} \wedge b_{j-1} \wedge (\bigwedge_{k=j}^{n} \Box^{k-j} q_k b_k) \wedge (\bigwedge_{k=j-1}^{n-1} \Box^{k-j+1}(b_{k-1} \wedge b_k \rightarrow \Box b_k))$$
$$\wedge \Box^{n-j+1}((b_{n-1} \wedge b_n) \rightarrow c) \models r_j ... r_n c \quad (4.2)$$

if and only if one of the following holds:

(a) $q_j = \Diamond$ and $r_j = \Box$ and

$$b_{j-1} \wedge (\bigwedge_{k=j+1}^{n} \Box^{k-j-1} q_k b_k) \wedge (\bigwedge_{k=j}^{n-1} \Box^{k-j}((b_{k-1} \wedge b_k) \rightarrow \Box b_k)) \wedge \Box^{n-j}((b_{n-1} \wedge b_n) \rightarrow c)$$
$$\models r_{j+1} ... r_n c$$

(b) $q_j = r_j$ and

$$b_{j-1} \wedge b_j \wedge (\bigwedge_{k=j+1}^{n} \Box^{k-j-1} q_k b_k) \wedge (\bigwedge_{k=j}^{n-1} \Box^{k-j}((b_{k-1} \wedge b_k) \rightarrow \Box b_k)) \wedge \Box^{n-j}((b_{n-1} \wedge b_n) \rightarrow c) \models r_{j+1}...r_n c$$

We will first show that the entailment in (a) doesn't hold. Consider the model $\mathfrak{M} = \langle \mathcal{W}, \mathcal{R}, v \rangle$ defined as follows:

- $\mathcal{W} = \{w_j, ..., w_n\}$
- $\mathcal{R} = \{(w_j, w_{j+1}), ..., (w_{n-1}, w_n)\}$
- $v(c, w) = false$ for all $w \in \mathcal{W}$
- for $w \neq w_j$: $v(b_k, w) = true$ if and only if $w = w_k$
- $v(b_k, w_j) = true$ if and only if $k = j - 1$

Notice that since each world (excepting w_n) has exactly one successor, the $\Box$- and $\Diamond$- quantifiers have the same behavior (except at w_n). It can easily be verified that $\mathfrak{M}, w_j$ satisfies the left-hand side of the above entailment for any tuple $q_{j+1}...q_n$: we have $\mathfrak{M}, w_j \models b_{j-1}$ by definition, we have $\mathfrak{M}, w_j \models \bigwedge_{k=j+1}^{n} \Box^{k-j-1} q_k\, b_k$ because $\mathfrak{M}, w_k \models b_k$ for $k \neq j$, we have $\mathfrak{M}, w_j \models \bigwedge_{k=j}^{n-1} \Box^{k-j}((b_{k-1} \wedge b_k) \rightarrow \Box b_k))$ since $\mathfrak{M}, w_j \not\models b_j$ and $\mathfrak{M}, w_k \not\models b_{k-1}$ for $k \neq j$, and finally we have $\mathfrak{M}, w_j \models \Box^{n-j}((b_{n-1} \wedge b_n) \rightarrow c)$ since $w_n \not\models b_{n-1}$. However, the right-hand side $r_{j+1}...r_n c$ is not satisfied at w_j: the only world accessible from w_j in $n - j - 1$ steps is w_n which does not satisfy c.

We have just shown that case (a) cannot hold, which means that Equation (4.2) holds if and only if (b) does. But if we apply the induction hypothesis to the entailment in (b), we find that it holds just in the case that $q_{j+1}...q_n = r_{j+1}...r_n$. It follows then that Equation (4.2) if and only if $q_j...q_n = r_j = r_n$, as desired. This completes our proof of the above statement.

We are now proceed to the proof of the lemma. By Theorem 2.3.1

$$\Box q_1(b_0 \wedge b_1) \wedge (\bigwedge_{k=2}^{n} \Box^k q_k b_k) \wedge (\bigwedge_{k=1}^{n-1} \Box^{k+1}\,(\,(b_{k-1} \wedge b_k) \rightarrow \Box b_k\,) \wedge \Box^{n+1}\,(\,(b_{n-1} \wedge b_n) \rightarrow c\,) \models \Box r_1...r_n c$$

holds just in the case that

$$q_1(b_0 \wedge b_1) \wedge (\bigwedge_{k=2}^{n} \Box^{k-1} q_k b_k) \wedge \bigwedge_{k=1}^{n-1} \Box^k\,(\,(b_{k-1} \wedge b_k) \rightarrow \Box b_k\,) \wedge \Box^n\,(\,(b_{n-1} \wedge b_n) \rightarrow c\,) \models r_1...r_n c$$

which in turn holds if and only if one of the following statements holds:

(i) $q_1 = \Diamond$ and $r_1 = \Box$ and

$$(\bigwedge_{k=2}^{n} \Box^{k-2} q_k b_k) \wedge (\bigwedge_{k=1}^{n-1} \Box^{k-1}((b_{k-1} \wedge b_k) \rightarrow \Box b_k)) \wedge \Box^{n-1}((b_{n-1} \wedge b_n) \rightarrow c) \models r_2 ... r_n c$$

(ii) $q_1 = r_1$ and

$$b_0 \wedge b_1 \wedge (\bigwedge_{k=2}^{n} \Box^{k-2} q_k b_k) \wedge (\bigwedge_{k=1}^{n-1} \Box^{k-1}((b_{k-1} \wedge b_k) \rightarrow \Box b_k)) \wedge \Box^{n-1}((b_{n-1} \wedge b_n) \rightarrow c) \models r_2 ... r_n c$$

We remark that if we set $j = 1$ in (a) above, then the left-hand-side of the entailment in (i) is logically weaker than that in (a), and the right-hand side matches that in (a). As we have already shown that the entailment in (a) does not hold, it follows that the entailment in (i) cannot hold either. Thus, we find that the desired entailment relation in the statement of the lemma holds if and only if (ii) does. This completes the proof since we have already shown in the induction above that the entailment in (ii) holds if and only if $q_2 ... q_n = r_2 ... r_n$, i.e. (ii) is true just in the case that $q_1 ... q_n = r_1 = r_n$. □

Lemma 4.1.11.
There is no clause w.r.t. ***D1*** *which is equivalent to λ and of strictly smaller size than λ.*

Proof. Let λ' be a **D1**-clause which is equivalent to λ. Suppose furthermore that λ' is the smallest such clause. As λ is non-tautologous and contains only $\Box$-literals as disjuncts, it follows that every disjunct of λ' must be either unsatisfiable or a $\Box$-literal (cf. Theorem 2.3.2). But since λ is satisfiable, so too is λ'. That means that if λ' contained an unsatisfiable disjunct, it could be removed to yield an equivalent but shorter clause, contradicting our assumption that λ' is of minimal length. Thus, λ' contains only $\Box$-literals.

Since $\lambda' \models \lambda$, every disjunct $\Box l$ of λ' must imply some disjunct $\Box q_1 ... q_n c$ of λ. Also, every disjunct $\Box l$ of λ' must be implied by some disjunct $\Box q_1 ... q_n c$ of λ, since otherwise we could remove $\Box l$ from λ' while preserving the equivalence between λ and λ'.

It follows then that each disjunct of λ' is implied by some disjunct of λ and implies some disjunct of λ. But since the disjuncts of λ do not imply each other (because of Lemma 4.1.9), it follows that each disjunct of λ' is equivalent to some disjunct of λ, and moreover that every disjunct of λ is equivalent to some disjunct of λ'.

This completes the proof since it is clear that the disjuncts $\Box q_1...q_nc$ of λ cannot be more compactly represented. Our proof works equally well for **D2**, since every **D2**-clause is also a **D1**-clause. □

Proof of Theorem 4.1.8. We begin with definition **D1**. Let λ and φ be as defined on page 95. We begin by distributing $\vee$ over $\wedge$ in order to transform φ into an equivalent disjunction of terms w.r.t. definition **D4**:

$$\varphi \equiv \bigvee_{(q_1,...,q_n)\in\{\Box,\Diamond\}^n} T_{q_1,...,q_n}$$

where $T_{q_1,...,q_n}$ is equal to

$$\Box q_1(b_0 \wedge b_1) \wedge (\bigwedge_{i=2}^{n} \Box^i q_i b_i) \wedge \bigwedge_{i=1}^{n-1} \Box^{i+1} (\,(b_{i-1} \wedge b_i) \rightarrow \Box b_i\,) \wedge \Box^{n+1} (\,(b_{n-1} \wedge b_n) \rightarrow c\,)$$

By Lemma 4.1.10, $T_{q_1,...,q_n} \models \Box q_1...q_nc$, and hence $T_{q_1,...,q_n} \models \lambda$. We thus have $\varphi \models \lambda$.

We now show that there is no stronger **D1**-clause w.r.t. which is implied by φ. Let λ' be a **D1**-clause such that $\varphi \models \lambda' \models \lambda$, and assume without loss of generality that λ' has no unsatisfiable disjuncts. As λ is a non-tautologous disjunction of $\Box$-literals, we know from Lemma 2.3.2 that every disjunct of λ' must be of the form $\Box l$ where l is a **D1**-clause such that $l \models r_1...r_nc$ for some quantifier string $r_1...r_n$. But according to Lemma 4.1.9, l must be equivalent either to $r_1...r_nc$ or to $r_1...r_k\bot$ for some $1 \leq k \leq n$ or $\bot$. It follows that λ' is equivalent to a clause having only disjuncts of the forms $\Box r_1...r_nc$ or $\Box r_1...r_k\bot$ $(1 \leq k \leq n)$ or $\Box\bot$.

As $\varphi \models \lambda'$, it must be the case that each of the terms $T_{q_1,...,q_n}$ implies λ', or equivalently $T_{q_1,...,q_n} \wedge \neg\lambda' \models \bot$. As we have shown above that the disjuncts of λ' are all $\Box$-literals, it follows from Theorem 2.3.1 that each term implies some disjunct of λ'. Moreover, we know from the preceding paragraph that the disjuncts of λ' to be of the forms $\Box r_1...r_nc$ or $\Box r_1...r_k\bot$ (where $r_i \in \{\Box, \Diamond\}$ and $1 \leq k \leq n$) or $\Box\bot$, so every $T_{q_1,...,q_n}$ must imply a formula with one of these forms.

We first show that a term $T_{q_1,...,q_n}$ cannot imply a formula of the form $\Box r_1...r_k\bot$ or $\Box\bot$. The proof is quite straightforward: we consider the model M with a single world w such that w is connected to itself and all propositional variables are true at w. It is not hard to see that $M, w \models T_{q_1,...,q_n}$ (for any tuple $(q_1, ..., q_n) \in \{\Box, \Diamond\}^n$). However, the formula $\Box r_1...r_k\bot$ (or $\Box\bot$) cannot hold at w since there is no world in M which has no successors.

Since a term $T_{q_1,...,q_n}$ cannot imply a formula $\Box r_1...r_k\bot$ or $\Box\bot$, it must be the case that $T_{q_1,...,q_n}$ implies some disjunct $\Box r_1...r_nc$ of λ'. By Lemma 4.1.10, the only

formula of this type which is implied by $T_{q_1,...,q_n}$ is the formula $\Box q_1...q_n c$. This means that for every tuple of quantifiers $(q_1, ..., q_n)$, there is a disjunct of λ' which is equivalent to $\Box q_1...q_n c$. It follows that every disjunct of λ is equivalent to some disjunct in λ', giving us $\lambda \models \lambda'$. We can thus conclude that λ is a prime implicate of φ.

This completes the proof, since we have already shown in Lemma 4.1.11 that there is no shorter **D1**-clause which is equivalent to λ than λ itself.

The above proof also works for definition **D2** since every clause w.r.t. **D2** is also a clause w.r.t. **D1**. In particular this means that any **D2**-clause which is a prime implicate w.r.t. **D1** is also a prime implicate w.r.t. **D2**, and that any **D2**-clause which is shortest among all equivalent **D1**-clauses is also shortest among **D2**-clauses. □

4.1.5 Bounds on the number of prime implicates

An examination of the set of candidate prime implicates constructed by our algorithm allows us to place a bound on the maximal number of non-equivalent prime implicates a formula can possess.

Theorem 4.1.12.
The number of non-equivalent prime implicates of a formula is at most double exponential in the length of the formula.

Proof. Consider some formula φ. We assume without loss of generality that φ is in NNF. We know from Theorem 4.1.4 that every prime implicate of φ is equivalent to some clause returned by **GenPI**. Every such clause is of the form $\bigvee_{T \in \mathbf{Dnf}(\varphi)} \theta_T$ where $\theta_T \in \Delta(T)$. As there can be at most $2^{|\varphi|}$ terms in $\mathbf{Dnf}(\varphi)$ by Theorem 2.4.6, these clauses can have no more than $2^{|\varphi|}$ disjuncts. Moreover, there are at most $|\varphi|$ choices for each disjunct θ_T since the cardinality of $\Delta(T)$ is bounded above by the number of conjuncts of T, which we know from Theorem 2.4.6 to be no more than $|\varphi|$. It follows then that there are at most $|\varphi|^{2^{|\varphi|}}$ clauses returned by **GenPI**, hence at most $|\varphi|^{2^{|\varphi|}}$ non-equivalent prime implicates of φ.

For a tighter bound, suppose that $\mathbf{Nnf}(\varphi)$ has at most l mutually non-equivalent literal subformulae appearing outside the scope of modal operators. We have already seen in the proof of Theorem 4.1.6 that this means that every prime implicate is equivalent to a clause $\bigvee_{T \in \mathcal{T}'} \theta_T$ for some subset $\mathcal{T}' \subseteq \mathcal{T}$ with cardinality at most 2^l. We also know that the elements in $\mathcal{T}$, and hence in the subset $\mathcal{T}'$, all have at most l non-equivalent conjuncts (by Theorem 2.4.6), which means that there are

at most l choices for each θ_T. It follows that there can be no more than l^{2^l} distinct equivalence classes of prime implicates of the formula φ. □

We can show this bound to be optimal.

Theorem 4.1.13.
The number of non-equivalent prime implicates of a formula may be double exponential in the length of the formula.

Proof. Let n be some natural number, and let a_{11}, a_{12}, ..., a_{n1}, a_{n2}, b_{11}, b_{12}, b_{12}, ..., b_{n1}, b_{n2} be distinct propositional variables. Consider the formula φ defined as

$$\bigwedge_{i=1}^{n}((\Diamond a_{i1} \wedge \Box b_{i1}) \vee (\Diamond a_{i2} \wedge \Box b_{i2}))$$

It is not hard to see that there will be 2^n terms in $\mathbf{Dnf}(\varphi)$, corresponding to the 2^n ways of deciding for each $i \in \{1, ..., n\}$ whether to take the first or second disjunct. Each term $T \in \mathbf{Dnf}(\varphi)$ will be of the form

$$\bigwedge_{i=1}^{n}(\Diamond a_{i\, f(i,T)} \wedge \Box b_{i\, f(i,T)})$$

where $f(i,T) \in \{1,2\}$ for all i. For each T, denote by $\mathcal{D}(T)$ the set of formulae $\{\Diamond(a_{\,f(i,T)} \wedge b_{1\, f(1,T)} \wedge ... \wedge b_{n\, f(n,T)})) \,|\, 1 \leq i \leq n\}$. Now consider the set of clauses $\mathcal{C}$ defined as

$$\{ \bigvee_{T \in \mathbf{Dnf}(\varphi)} d_T \,|\, d_T \in \mathcal{D}(T)\}$$

Notice that there are n^{2^n} clauses in $\mathcal{C}$ since each clause corresponds to a choice of one of the n elements in $\mathcal{D}(T)$ for each of the 2^n terms T in $\mathbf{Dnf}(\varphi)$. This number is double exponential in $|\varphi|$ since the length of φ is linear in n. In order to complete the proof, we show that (i) all of the clauses in $\mathcal{C}$ are prime implicates of φ and (ii) that the clauses in $\mathcal{C}$ are mutually non-equivalent.

We begin by showing that $\lambda_1 \not\models \lambda_2$ for every pair of distinct elements λ_1 and λ_2 in $\mathcal{C}$. This immediately gives us (ii) and will prove useful in the proof of (i). Let λ_1 and λ_2 be distinct clauses in $\mathcal{C}$. As λ_1 and λ_2 are distinct, there must be some term $T \in \mathbf{Dnf}(\varphi)$ for which λ_1 and λ_2 choose different elements from $\mathcal{D}(T)$. Let d_1 be the element from $\mathcal{D}(T)$ appearing as a disjunct in λ_1, let d_2 be the element in $\mathcal{D}(T)$ which is a disjunct in λ_2, and let $a_{j,k}$ be the a-literal which appears in d_2 (and hence not in d_1). Consider the formula $\rho = \Box(\neg a_{j,k} \wedge \neg b_{1,k_1} \wedge ... \wedge \neg b_{n,k_n})$, where the tuple $(k_1, ..., k_n)$ is just like the tuple associated with T except that the 1's and 2's are inversed. Clearly $d_1 \wedge \rho$ is consistent, since the variables in ρ do not

appear in d_1. But ρ is inconsistent with every disjunct in λ_2, since by construction every disjunct in λ_2 contains a literal whose negation appears in ρ. It follows that $\lambda_2 \models \neg\rho$ but $\lambda_1 \not\models \neg\rho$, and hence $\lambda_1 \not\models \lambda_2$.

We now prove (i). Let λ be a clause in $\mathcal{C}$, and let π be a prime implicate of φ which implies λ. By Theorem 4.1.4, we know that π must be equivalent to one of the clauses output by **GenPI**, and more specifically to a clause output by **GenPI** which is a disjunction of $\Diamond$-literals (because of Theorem 2.3.2). We remark that the set $\mathcal{C}$ is composed of exactly those candidate clauses which are disjunctions of $\Diamond$-literals, so π must be equivalent to some clause in $\mathcal{C}$. But we have just shown that the only element in $\mathcal{C}$ which implies λ is λ itself. It follows that $\pi \equiv \lambda$, which means that λ is a prime implicate of φ. □

This worst-case result is robust in that it can be improved neither by restricting the depth of the input formulae, nor by using less expressive notions of prime implicate, as is demonstrated by the following theorem.

Theorem 4.1.14.
*If prime implicates are defined using either **D1** or **D2**, then the number of non-equivalent prime implicates of a formula may be doubly exponential in the length of the formula.*

Proof. Let λ and φ be as defined on page 95. Set φ' equal to the formula obtained from φ by replacing c in the last conjunct of φ by $c \wedge d$. Set Σ equal to the set of clauses that can be obtained from λ by replacing zero or more occurrences of c by d. For example, if $n = 1$, then $\Sigma = \{\Box\Diamond c \vee \Box\Box c, \Box\Diamond d \vee \Box\Box c, \Box\Diamond c \vee \Box\Box d, \Box\Diamond d \vee \Box\Box d\}$. There are 2^{2^n} elements in Σ since we choose for each of the 2^n disjuncts of λ whether to change c into d. We intend to show that the clauses in Σ are all pairwise non-equivalent prime implicates of φ'. The proof that every element in Σ is indeed a prime implicate of φ' (w.r.t. both **D1** and **D2**) proceeds quite similarly to the proof that λ is a prime implicate (w.r.t. **D1** and **D2**) of φ (see proof of Theorem 4.1.8), so we will not repeat it here. Instead we will show that all of the elements in Σ are pairwise non-equivalent. To do so, we consider any two distinct elements α and β of Σ. Since α and β are distinct, there must be some string of quantifiers $q_1...q_n$ such that α has a disjunct $\Box q_1...q_n\gamma$ ($\gamma \in \{c, d\}$) which is not a disjunct of β. Now if $\alpha \models \beta$, then we would have $\Box q_1...q_n\gamma \models \beta$, and hence $\Box q_1...q_n\gamma \models \Box r_1...r_n\zeta$ for some disjunct $r_1...r_n\zeta$ of β. But by using Lemma 4.1.9, we see that this can only happen if $r_1...r_n = q_1...q_n$ and $\gamma = \zeta$, i.e. if $\Box q_1...q_n\gamma$ is a disjunct of β. This is a contradiction, so we must have $\alpha \not\models \beta$. It follows that the elements of Σ are pairwise non-equivalent, and hence that φ' possesses a double exponential number of prime implicates. □

Theorems 4.1.8 and 4.1.14 suggest that definitions **D1** and **D2** do not yield especially interesting approximate notions of prime implicate, as they induce a significant loss of expressivity without any improvement in the size or number of prime implicates in the worst-case.

Remark 4.1.15.
The double-exponential lower bounds in this section are pretty frightening, but it should be noted that this kind of result is not altogether uncommon for logics of this expressivity. For example, in [DTR06], it is shown that if one approximates an $\mathcal{ALC}$ concept by an $\mathcal{ALE}$ concept, the approximated concept may be double-exponentially larger than the original.

4.1.6 Improving the efficiency of GenPI

Our generation algorithm **GenPI** corresponds to the simplest possible implementation of the **Distribution** property, and it is quite clear that it does not represent a practicable way for producing prime implicates. One major source of inefficiency is the high number of clauses that are generated, so if we want to design a more efficient algorithm, we need to find ways to generate fewer candidate clauses. There are a couple of different techniques that could be used. One very simple method which could yield a smaller number of clauses is to eliminate from $\Delta(T)$ those elements which are not prime implicates of T, thereby decreasing the cardinalities of the $\Delta(T)$ and hence of CANDIDATES. To do this, we simply test whether any of $\Box$-formulae in $\Delta(T)$ is a tautology (and remove any discovered tautologies from $\Delta(T)$) and then compare the $\Diamond$-literals in $\Delta(T)$, discarding any weaker elements. If we apply this technique to Example 4.1.1, we would remove $\Diamond b$ from $\Delta(T_1)$, thereby reducing the cardinality of CANDIDATES from 20 to 15.

More substantial savings could be achieved by using a technique developed in the framework of propositional logic (cf. discussion in [Mar00]) which consists in calculating the prime implicates of T_1, then the prime implicates of $T_1 \vee T_2$, then those of $T_1 \vee T_2 \vee T_3$, and so on until we get the prime implicates of the full disjunction of terms. By interleaving comparison and construction, we can eliminate early on a partial clause that cannot give rise to prime implicates instead of producing all of the extensions of the partial clause and then deleting them one by one during the comparison phase. In Example 4.1.1, there were only two terms, but imagine that there was a third term T_3. Then by applying this technique, we would first produce the 5 prime implicates of $T_1 \vee T_2$ and then we would compare the $5|\Delta(T_3)|$ candidate clauses of $T_1 \vee T_2 \vee T_3$. Compare this with the current algorithm which generates and then compares $20|\Delta(T_3)|$ candidate clauses.

Another very simple technique which can be fruitfully combined with the previous one is the following: before forming the different disjunctions of prime implicates of $T_1 \vee ... \vee T_{k-1}$ and elements from $\Delta(T_k)$, we first check to see whether any prime implicates of $T_1 \vee ... \vee T_{k-1}$ are already implied by T_k; any prime implicate fulfilling this condition is added directly to the set of prime implicates of $T_1 \vee ... \vee T_k$ and is not used in the construction of candidate prime implicates. The use of this technique helps reduce the number of candidate prime implicates of $T_1 \vee ... \vee T_k$ which need to be considered, and it also yields prime implicates which have fewer unnecessary disjuncts. We illustrate the previous two techniques on the following example:

Example 4.1.16.
We would like to generate the prime implicates of the formula φ defined as:

$$\begin{aligned} & (a \vee \Box_2\Box_1 d) \\ \wedge \quad & (\neg a \vee \Diamond_2(a \wedge b)) \\ \wedge \quad & \Box_1((a \vee \Diamond_2\top) \wedge (\Box_2\bot \vee c)) \\ \wedge \quad & \Diamond_1(a \wedge \neg c \wedge \Box_2\bot) \end{aligned}$$

We begin by using **Dnf** to rewrite φ as an equivalent disjunction of satisfiable terms. We obtain the following three satisfiable terms (a fourth unsatisfiable term is eliminated):

$$T_1 = a \wedge \Diamond_2(a \wedge b) \wedge \Box_1((a \vee \Diamond_2\top) \wedge (\Box_2\bot \vee c)) \wedge \Diamond_1(a \wedge \neg c \wedge \Box_2\bot)$$
$$T_2 = \neg a \wedge \Box_2\Box_1 d \wedge \Box_1((a \vee \Diamond_2\top) \wedge (\Box_2\bot \vee c)) \wedge \Diamond_1(a \wedge \neg c \wedge \Box_2\bot)$$
$$T_3 = \Diamond_2(a \wedge b) \wedge \Box_2\Box_1 d \wedge \Box_1((a \vee \Diamond_2\top) \wedge (\Box_2\bot \vee c)) \wedge \Diamond_1(a \wedge \neg c \wedge \Box_2\bot)$$

We now generate the prime implicates of the first term T_1. To do so, we compute the set $\Delta(T_1)$ which is:

$$\{a, \Diamond_2(a \wedge b), \Box_1((a \vee \Diamond_2\top) \wedge (\Box_2\bot \vee c)), \Diamond_1(a \wedge \neg c \wedge \Box_2\bot \wedge (a \vee \Diamond_2\top) \wedge (\Box_2\bot \vee c))\}$$

As the four elements in $\Delta(T_1)$ are all mutually non-implying, they are all prime implicates of T_1.

We next want to compute the prime implicates of the disjunction $T_1 \vee T_2$. For this, we must compute the disjunctions of the prime implicates of T_1 (just computed) with elements in the set $\Delta(T_2) =$

$$\{\neg a, \Box_2\Box_1 d, \Box_1((a \vee \Diamond_2\top) \wedge (\Box_2\bot \vee c)), \Diamond_1(a \wedge \neg c \wedge \Box_2\bot \wedge (a \vee \Diamond_2\top) \wedge (\Box_2\bot \vee c))\}$$

We notice, however, that the prime implicates $\Box_1((a \vee \Diamond_2\top) \wedge (\Box_2\bot \vee c))$ and $\Diamond_1(a \wedge \neg c \wedge \Box_2\bot \wedge (a \vee \Diamond_2\top) \wedge (\Box_2\bot \vee c))$ of T_1 are both implied by T_2, so they

are also prime implicates of $T_1 \vee T_2$. We can thus remove from consideration any disjunctions built using these prime implicates. By combining the remaining two prime implicates of T_1 with the elements in $\Delta(T_2)$, we obtain 8 candidate prime implicates. The candidate $a \vee \neg a$ is a tautology and hence is removed in the comparison phase. The three candidates which have either $\Box_1((a \vee \Diamond_2 \top) \wedge (\Box_2 \bot \vee c))$ or $\Diamond_1(a \wedge \neg c \wedge \Box_2 \bot \wedge (a \vee \Diamond_2 \top) \wedge (\Box_2 \bot \vee c))$ as a disjunct will also be removed as they are logically weaker than the two prime implicates already identified. After elimination of weaker elements, there are three remaining candidates $a \vee \Box_2 \Box_1 d$, $\Diamond_2(a \wedge b) \vee \neg a$, and $\Diamond_2(a \wedge b) \vee \Box_2 \Box_1 d$, which are all prime implicates of $T_1 \vee T_2$.

Finally, we are ready to compute the prime implicates of the entire disjunction $T_1 \vee T_2 \vee T_3$. Normally this would involve disjoining the prime implicates of $T_1 \vee T_2$ with the elements in the set $\Delta(T_3)$. However, we remark that each of the prime implicates of $T_1 \vee T_2$ is implied by T_3, so the prime implicates of $\varphi \equiv T_1 \vee T_2 \vee T_3$ are exactly the same as those of $T_1 \vee T_2$ (as one would expect since $T_1 \vee T_2 \equiv T_1 \vee T_2 \vee T_3$).

We thus find that φ has 5 prime implicates (modulo equivalence), which are: $a \vee \Box_2 \Box_1 d$, $\Diamond_2(a \wedge b) \vee \neg a$, and $\Diamond_2(a \wedge b) \vee \Box_2 \Box_1 d$, $\Box_1((a \vee \Diamond_2 \top) \wedge (\Box_2 \bot \vee c))$ and $\Diamond_1(a \wedge \neg c \wedge \Box_2 \bot \wedge (a \vee \Diamond_2 \top) \wedge (\Box_2 \bot \vee c))$.

Given that the number of elements in CANDIDATES can be double exponential in the length of the input formula, another strategy for improving the efficiency our algorithm is to try to reduce the size of the input. For instance, a natural idea would be to break conjunctions of formulae into their conjuncts, and then calculate the prime implicates of each of the conjuncts. Unfortunately, however, we cannot apply this method to every formula as the prime implicates of the conjuncts are not necessarily prime implicates of the full conjunction. One solution which was proposed in the context of approximation of description logic concepts (cf. [BT02], [TB07]) is to identify simple syntactic conditions that guarantee that we will get the same result if we break the formula into its conjuncts. For instance, one possible condition is that the conjuncts do not share any propositional variables. The formula φ from Example 4.1.1 satisfies this condition since the signatures of the conjuncts a, $(\Diamond_1 (b \wedge c) \wedge \Diamond_1 b) \vee (\Diamond_1 b \wedge \Diamond_1 (c \vee d) \wedge \Box_1 e \wedge \Box_1 f)$ and $\Box_2 \bot$ are all disjoint. By generating the prime implicates of the conjuncts separately, we can directly identify the prime implicates a and $\Box_2 \bot$, and we only have 6 candidate clauses of $(\Diamond_1 (b \wedge c) \wedge \Diamond_1 b) \vee (\Diamond_1 b \wedge \Diamond_1 (c \vee d) \wedge \Box_1 e \wedge \Box_1 f)$ to compare. If we also remove weaker elements from the $\Delta(T_i)$ as suggested above, we get only 3 candidate clauses for $(\Diamond_1 (b \wedge c) \wedge \Diamond_1 b) \vee (\Diamond_1 b \wedge \Diamond_1 (c \vee d) \wedge \Box_1 e \wedge \Box_1 f)$, all of which are prime implicates of φ.

Another important source of inefficiency in our algorithm is the comparison phase in which we compare all candidate clauses one-by-one in order to identify

the strongest ones. The problem with this is of course that in the worst-case there can be a double exponential number of candidate clauses, simply because there may be double-exponentially many distinct prime implicates, and each prime implicate must be equivalent to some candidate clause. Keeping all of these double-exponentially many clauses in memory will generally not be feasible. Fortunately, however, it is not necessary to keep the all of the candidate clauses in memory at once since we can generate them on demand from the sets $\Delta(T)$. Indeed, as we demonstrate in the following theorem, by implementing our algorithm in a more clever fashion, we obtain an algorithm which outputs the prime implicates iteratively while requiring only single exponential space (the output of the algorithm could of course be double-exponentially large because of Theorem 4.1.13).

Theorem 4.1.17.
There exists an algorithm which runs in single exponential space in the size of the input and incrementally outputs, without duplicates, the set of prime implicates of the input formula.

Proof. Let the sets $\mathcal{T}$ and CANDIDATES and the function Δ be defined as in Figure 4.1. We assume that $\mathcal{T}$ is ordered: $\mathcal{T} = \{T_1, ..., T_n\}$. For each $T_i \in \mathcal{T}$, we let max_i denote the number of elements in $\Delta(T_i)$, and we assume an ordering on the elements of $\Delta(T_i)$: $\Delta(T_i) = \{\tau_{i,1}, ..., \tau_{i,max_i}\}$. Notice that the tuples in $\{1, .., max_1\} \times ... \times \{1, ..., max_n\}$ can be ordered using the standard lexicographic ordering $<_{lex}$: $(a_1, ..., a_n) <_{lex} (b_1, ..., b_n)$ if and only if there is some $1 \leq j \leq n$ such that $a_j < b_j$ and $a_k \leq b_k$ for all $1 \leq k \leq j-1$. Now set $maxindex = \Pi_{i=1}^{n} max_i$, and let $f : \{1, .., max_1\} \times ... \times \{1, ..., max_m\} \rightarrow \{1, ..., maxindex\}$ be the bijection defined as follows: $f(a_1, ..., a_n) = m$ if and only if $(a_1, ..., a_n)$ is the m-th tuple in the lexicographic ordering of $\{1, .., max_1\} \times ... \times \{1, ..., max_n\}$. We will denote by λ_m the unique clause of form $\tau_{1,a_1} \vee ... \vee \tau_{n,a_n}$ such that $f(a_1, ..., a_n) = m$. We remark that given an index $m \in \{1, ..., maxindex\}$ and the sets $\Delta(T_1)$, ..., $\Delta(T_n)$, it is possible to generate in polynomial space (in the size of the sets $\Delta(T_1)$, ..., $\Delta(T_n)$) the clause λ_m. We make use of this fact in our modified version of algorithm **GenPI**, which is obtained from **GenPI** by replacing Steps 3 and 4 with the following:

(3') For $i = 1$ to $maxindex$: if **Entails**(λ_j, λ_i)=**no** for all $j < i$ *and* either **Entails**(λ_j, λ_i)=**no** or **Entails**(λ_i, λ_j)=**yes** for every $i < j \leq maxindex$, then output λ_i.

The proofs of termination, correctness, and completeness of this modified version of **GenPI** are very similar to corresponding results for the original version

(Theorem 4.1.4), so we will omit the details. We will instead focus on the modified algorithm's spatial complexity. The first step clearly runs in single-exponential space in $|\varphi|$, since deciding the satisfiability of φ takes only polynomial space in $|\varphi|$, and generating the elements in $\mathbf{Dnf}(\varphi)$ takes at most single exponential space in $|\varphi|$ (refer to Theorems 2.4.6 and 2.4.7). Step 2 also uses no more than single exponential space in $|\varphi|$, since each of the sets $\Delta(T)$ associated with a term $T_i \in \mathcal{T}$ has polynomial size in T_i. Finally, for Step 3', we use the above observation that the generation of a given λ_i from its index i can be done in polynomial space in the size of the sets $\Delta(T_1)$, ..., $\Delta(T_n)$, and hence in single exponential space in $|\varphi|$. This is sufficient since for the comparisons in Step 3', we only need to keep two candidate clauses in memory at any one time, and deciding whether one candidate clause entails another can be accomplished in single-exponential space since both clauses have single exponential size in $|\varphi|$, and entailment in $\mathcal{K}_n$ can be done in polynomial space in the size of the input formulae (Corollary 2.5.9). □

Although the modified algorithm outlined in the proof of Theorem 4.1.17 has a much better spatial complexity than the original, it still does not yield a practicable means for generating prime implicates. The reason is that we still need to compare each of the candidate clauses against all the other candidate clauses in order to decide whether a candidate clause is a prime implicate or not. Given that there may be double-exponentially many candidate clauses, this means that our algorithm may need to perform double-exponentially many entailment tests before producing even a single prime implicate. A much more promising approach would be to test directly whether or not a candidate clause is a prime implicate *without considering all of the other candidate clauses*. In order to implement such an approach, we must come up with a procedure for determining whether or not a given clause is a prime implicate. This will be our objective in the following section.

4.2 Prime Implicate Recognition

The focus of this section is the problem of recognizing prime implicates, that is, the problem of deciding whether a given clause is a prime implicate of a given formula. This problem is of central importance, as any algorithm for generating prime implicates must contain (implicitly or explicitly) some mechanism for ensuring that the generated clauses are indeed prime implicates.

4.2.1 Lower bound

In propositional logic, prime implicate recognition is BH_2-complete [Mar00], being as hard as both satisfiability and deduction. In $\mathcal{K}_n$, satisfiability, unsatisfiability, and deduction are all PSPACE-complete, so we cannot hope to find a prime implicate recognition algorithm with a complexity of less than PSPACE.

Theorem 4.2.1.
Prime implicate recognition is PSPACE*-hard.*

Proof. The reduction is simple: a formula φ is unsatisfiable if and only if $\bot$ is a prime implicate of φ. This suffices as the problem of checking the unsatisfiability of formulae in $\mathcal{K}_n$ is known to be PSPACE-complete. □

4.2.2 Naïve approach

In order to obtain a first upper bound, we can exploit Theorem 4.1.6 which tells us that there exists a polynomial function f such that the length of the smallest clausal representation of a prime implicate of a formula φ is bounded by $2^{f(|\varphi|)}$. This leads to a simple exponential-space procedure for determining if a clause λ is a prime implicate of a formula φ: we simply check for every clause λ' of length at most $2^{f(|\varphi|)}$ whether λ' is an implicate of φ which implies λ but is not implied by λ.

Theorem 4.2.2.
Prime implicate recognition is in EXPSPACE*.*

Proof. Consider the following algorithm for determining whether a clause λ is a prime implicate of φ: check for each clause λ' of length at most $2|\varphi| * 2^{|\varphi|} + (2^{|\varphi|} - 1)$ which is an implicate of φ whether both **Entails**(λ', λ)=**yes** and **Entails**(λ, λ')=**no**. If there is some λ' satisfying these conditions, return **no**, otherwise return **yes**. Notice that if this algorithm returns **yes**, then there is no implicate of length at most $2|\varphi| * 2^{|\varphi|} + (2^{|\varphi|} - 1)$ that is strictly stronger than λ, and hence by Theorem 4.1.6 no strictly stronger implicate of any length, making λ a prime implicate. If the algorithm returns **no**, then we have found a clause implied by φ which is strictly stronger than λ, so λ is not a prime implicate. The algorithm is thus both correct and complete. As the algorithm consists solely in testing the satisfiability and unsatisfiability of formulae having length at most single exponential in $|\varphi| + |\lambda|$, and both tasks can be accomplished in polynomial space in the size of the input (by Theorem 2.5.7), the algorithm can be executed in exponential space. □

4.2.3 Decomposition theorem

Of course, the problem with the naïve approach just presented is that it doesn't take into account the structure of λ, so we end up comparing a huge amount of irrelevant clauses, which is exactly what we were hoping to avoid. The algorithm that we propose later in this chapter avoids this problem by exploiting the information in the input formula and clause in order to cut down on the number of clauses to test. The key to our algorithm is the following theorem which shows us how the general problem of prime implicate recognition can be decomposed into the more specialized tasks of prime implicate recognition for propositional formulae, $\Box$-formulae, and $\Diamond$-formulae.

To simplify the presentation of the theorem, we let $\Pi(\varphi)$ refer to the set of prime implicates of φ, and we use the notation $\lambda \setminus \{l_1, ..., l_n\}$ to refer to the clause obtained by removing each of the literals l_i from λ. For example $(a \vee b \vee e \vee \Diamond c) \setminus \{a, \Diamond c\}$ refers to the clause $b \vee e$.

Theorem 4.2.3.
Let φ be a formula of $\mathcal{K}_n$, and let $\lambda =$

$$\gamma_1 \vee ... \vee \gamma_k \vee \bigvee_{i=1}^{n} (\Diamond_i \psi_{i,1} \vee ... \vee \Diamond_i \psi_{i,l_i} \vee \Box_i \chi_{i,1} \vee ... \vee \Box_i \chi_{i,m_i})$$

($\gamma_1 \vee ... \vee \gamma_k$ propositional) be a non-tautologous clause such that (a) $\chi_{i,j} \equiv \chi_{i,j} \vee \psi_{i,1} \vee ... \vee \psi_{i,l_i}$ for all $1 \leq i \leq n$ and $1 \leq j \leq m_i$, and (b) there is no literal l in λ such that $\lambda \equiv \lambda \setminus \{l\}$. Then $\lambda \in \Pi(\varphi)$ if and only if the following conditions hold:

1. $\gamma_1 \vee ... \vee \gamma_k \in \Pi(\varphi \wedge \neg(\lambda \setminus \{\gamma_1, ..., \gamma_k\}))$
2. $\Box_i (\chi_{i,j} \wedge \neg\psi_{i,1} \wedge ... \wedge \neg\psi_{i,l_i}) \in \Pi(\varphi \wedge \neg(\lambda \setminus \{\Box\chi_{i,j}\}))$
   for every $1 \leq i \leq n$ and $1 \leq j \leq m_i$
3. $\Diamond_i (\psi_{i,1} \vee ... \vee \psi_{i,l_i}) \in \Pi(\varphi \wedge \neg(\lambda \setminus \{\Diamond_i \psi_{i,1}, ..., \Diamond_i \psi_{i,l_i}\}))$
   for every $1 \leq i \leq n$

For the proof of Theorem 4.2.3, we will require the following two lemmas:

Lemma 4.2.4.
Let φ be a formula of $\mathcal{K}_n$, and let $\lambda =$

$$\gamma_1 \vee ... \vee \gamma_k \vee \bigvee_{i=1}^{n} (\Diamond_i \psi_{i,1} \vee ... \vee \Diamond_i \psi_{i,l_i} \vee \Box_i \chi_{i,1} \vee ... \vee \Box_i \chi_{i,m_i})$$

be a non-tautologous clause such that $\gamma_1 \vee ... \vee \gamma_k$ is a propositional clause. Suppose furthermore that there is no literal l in λ such that $\lambda \equiv \lambda \setminus \{l\}$. If $\gamma_1 \vee ... \vee \gamma_k \notin$

$\Pi(\varphi \wedge \neg(\lambda \setminus \{\gamma_1, ..., \gamma_k\}))$, *or* $\Diamond_i (\psi_{i,1} \vee ... \vee \psi_{i,l_i}) \notin \Pi(\varphi \wedge \neg(\lambda \setminus \{\Diamond_i \psi_{i,1}, ..., \Diamond_i \psi_{i,l_i}\}))$ *for some* i, *or* $\Box_i (\chi_{i,j} \wedge \neg\psi_{i,1} \wedge ... \wedge \neg\psi_{i,l_i}) \notin \Pi(\varphi \wedge \neg(\lambda \setminus \{\Box\chi_{i,j}\}))$ *for some* i *and* j, *then* $\lambda \notin \Pi(\varphi)$.

Proof. We will only consider the case where $\varphi \models \lambda$ because if $\varphi \not\models \lambda$ then we immediately get $\lambda \notin \Pi(\varphi)$.

Let us first suppose that $\gamma_1 \vee ... \vee \gamma_k \notin \Pi(\varphi \wedge \neg(\lambda \setminus \{\gamma_1, ..., \gamma_k\}))$. Since $\varphi \models \lambda$, we must also have $\varphi \wedge \neg(\lambda \setminus \{\gamma_1, ..., \gamma_k\}) \models \gamma_1 \vee ... \vee \gamma_k$, so $\gamma_1 \vee ... \vee \gamma_k$ is an implicate of $\varphi \wedge \neg(\lambda \setminus \{\gamma_1, ..., \gamma_k\})$. As $\gamma_1 \vee ... \vee \gamma_k$ is known not to be a prime implicate of $\varphi \wedge \neg(\lambda \setminus \{\gamma_1, ..., \gamma_k\})$, it follows that there must be some clause λ' such that $\varphi \wedge \neg(\lambda \setminus \{\gamma_1, ..., \gamma_k\}) \models \lambda' \models \gamma_1 \vee ... \vee \gamma_k \not\models \lambda'$. Now consider the clause $\lambda'' = \lambda' \vee \bigvee_{i=1}^{n} (\Diamond_i \psi_{i,1} \vee ... \vee \Diamond_i \psi_{i,l_i} \vee \Box_i\chi_{i,1} \vee ... \vee \Box_i\chi_{i,m_i})$. We know that $\varphi \models \lambda''$ since $\varphi \wedge \neg(\lambda \setminus \{\gamma_1, ..., \gamma_k\}) \models \lambda'$, and that $\lambda'' \models \lambda$ because $\lambda' \models \gamma_1 \vee ... \vee \gamma_k$. We also have $\lambda \not\models \lambda''$ since λ' must be equivalent to a propositional clause (by Theorem 2.3.2) and the propositional part of λ (namely $\gamma_1 \vee ... \vee \gamma_k$) does not imply λ'. It follows then that $\varphi \models \lambda'' \models \lambda \not\models \lambda''$, so $\lambda \notin \Pi(\varphi)$.

Next suppose that $\Diamond_s (\psi_{s,1} \vee ... \vee \psi_{s,l_s}) \notin \Pi(\varphi \wedge \neg(\lambda \setminus \{\Diamond_s \psi_{s,1}, ..., \Diamond_s \psi_{s,l_s}\}))$. Now $\Diamond_s (\psi_{s,1} \vee ... \vee \psi_{s,l_s})$ must be an implicate of $\varphi \wedge \neg(\lambda \setminus \{\Diamond_s \psi_{s,1}, ..., \Diamond\psi_{s,l_s}\})$ since we have assumed that $\varphi \models \lambda$. As $\Diamond_s (\psi_{s,1} \vee ... \vee \psi_{s,l_s})$ is not a prime implicate of $\varphi \wedge \neg(\lambda \setminus \{\Diamond_s \psi_{s,1}, ..., \Diamond\psi_{s,l_s}\})$, it follows that there is some λ' such that $\varphi \wedge \neg(\lambda \setminus \{\Diamond_s \psi_{s,1}, ..., \Diamond\psi_{s,l_s}\}) \models \lambda' \models \Diamond_s (\psi_{s,1} \vee ... \vee \psi_{s,l_s}) \not\models \lambda'$. Set λ'' equal to

$$\gamma_1 \vee ... \vee \gamma_k \vee \lambda' \vee \bigvee_{\substack{1 \leq i \leq n \\ i \neq s}} (\Diamond_i \psi_{i,1} \vee ... \vee \Diamond_i \psi_{i,l_i}) \vee \bigvee_{i=1}^{n} (\Box_i\chi_{i,1} \vee ... \vee \Box_i\chi_{i,m_i})$$

Because of Theorem 2.3.2, we know that λ' is a disjunction of $\Diamond$-literals, so according to Theorem 2.3.3 we must have $\lambda \not\models \lambda''$ since $\Diamond_s (\psi_{s,1} \vee ... \vee \psi_{s,l_s}) \not\models \lambda'$. We also know that $\varphi \models \lambda''$ since $\varphi \wedge \neg(\lambda \setminus \{\Diamond_s \psi_{s,1}, ..., \Diamond_s \psi_{s,l_s}\}) \models \lambda'$ and that $\lambda'' \models \lambda$ since $\lambda' \models \Diamond_s (\psi_{s,1} \vee ... \vee \psi_{s,l_s})$. That means that $\varphi \models \lambda'' \models \lambda \not\models \lambda''$, so $\lambda \notin \Pi(\varphi)$.

Finally consider the case where $\Box_s (\chi_{s,t} \wedge \neg\psi_{s,1} \wedge ... \wedge \neg\psi_{s,l_s}) \notin \Pi(\varphi \wedge \neg(\lambda \setminus \{\Box\chi_{s,t}\}))$. We know that $\varphi \models \lambda$ and hence that $\varphi \wedge \neg(\lambda \setminus \{\Box_s \chi_{s,t}\}) \models \Box_s \chi_{s,t}$. Moreover, since $\neg(\lambda \setminus \{\Box_s \chi_{s,t}\}) \models \neg\Diamond_s \psi_{s,j}$ for all $1 \leq j \leq s_l$, we have $\varphi \wedge \neg(\lambda \setminus \{\Box_s \chi_{s,t}\}) \models \Box_s (\chi_{s,t} \wedge \neg\psi_{s,1} \wedge ... \wedge \neg\psi_{s,l_s})$. Thus, if $\Box_s (\chi_{s,t} \wedge \neg\psi_{s,1} \wedge ... \wedge \neg\psi_{s,l_s}) \notin \Pi(\varphi \wedge \neg(\lambda \setminus \{\Box_s \chi_{s,t}\}))$, it must mean that there is some λ' such that $\varphi \wedge \neg(\lambda \setminus \{\Box_s \chi_{s,t}\}) \models \lambda' \models \Box_s (\chi_{s,t} \wedge \neg\psi_{s,1} \wedge ... \wedge \neg\psi_{s,l_s}) \not\models \lambda'$. By assumption, λ is not a tautology, so $\Box_s (\chi_{s,t} \wedge \neg\psi_{s,1} \wedge ... \wedge \neg\psi_{s,l_s})$ cannot be a tautology either. As $\lambda' \models \Box_s (\chi_{s,t} \wedge \neg\psi_{s,1} \wedge ... \wedge \neg\psi_{s,l_s})$ and $\Box_s (\chi_{s,t} \wedge \neg\psi_{s,1} \wedge ... \wedge \neg\psi_{s,l_s})$ is not a tautology, it follows from Theorem 2.3.2 that λ' is equivalent to some formula

$\Box_s \zeta_1 \vee ... \vee \Box_s \zeta_p$. Now let $\lambda'' =$

$$\gamma_1 \vee ... \vee \gamma_k \vee \bigvee_{i=1}^{n} (\Diamond_i \psi_{i,1} \vee ... \vee \Diamond_i \psi_{i,l_i}) \vee (\lambda' \vee \bigvee_{\substack{1 \leq j \leq m_s \\ j \neq t}} \Box_s \chi_{s,j}) \vee \bigvee_{\substack{1 \leq i \leq n \\ i \neq s}} (\Box_i \chi_{i,1} \vee ... \vee \Box_i \chi_{i,m_i})$$

As $\varphi \wedge \neg(\lambda \setminus \{\Box_s \chi_{s,t}\}) \models \Box_s \zeta_1 \vee ... \vee \Box_s \zeta_p$, it must be the case that $\varphi \models \lambda''$. Also, we know that there can be no q such that $\chi_{s,t} \models \zeta_q \vee \psi_{s,1} \vee ... \vee \psi_{s,l_s}$ because otherwise we would have $\chi_{s,t} \wedge \neg\psi_{s,1} \wedge ... \wedge \neg\psi_{s,l_s} \models \zeta_q$ and hence $\Box_s (\chi_{s,t} \wedge \neg\psi_{s,1} \wedge ... \wedge \neg\psi_{s,l_s}) \models \Box_s \zeta_1 \vee ... \vee \Box_s \zeta_p$. Similarly, there can be no $q \neq t$ such that $\chi_{s,t} \models \chi_{s,q} \vee \psi_{s,1} \vee ... \vee \psi_{s,l_s}$ because this would mean that $\lambda \equiv \lambda \setminus \{\Box\chi_{s,t}\}$, contradicting our assumption that there are no superfluous disjuncts in λ. It follows then by Theorem 2.3.3 that $\lambda \not\models \lambda''$. Thus, $\varphi \models \lambda'' \models \lambda \not\models \lambda''$, which means $\lambda \notin \Pi(\varphi)$. □

Lemma 4.2.5.
Let φ be a formula of $\mathcal{K}_n$, and let $\lambda =$

$$\gamma_1 \vee ... \vee \gamma_k \vee \bigvee_{i=1}^{n} (\Diamond_i \psi_{i,1} \vee ... \vee \Diamond_i \psi_{i,l_i} \vee \Box_i \chi_{i,1} \vee ... \vee \Box_i \chi_{i,m_i})$$

be a non-tautologous clause such that $\gamma_1 \vee ... \vee \gamma_k$ is a propositional clause. Suppose furthermore that there is no literal l in λ such that $\lambda \equiv \lambda \setminus \{l\}$. Then if $\lambda \notin \Pi(\varphi)$, either $\gamma_1 \vee ... \vee \gamma_k \notin \Pi(\varphi \wedge \neg(\lambda \setminus \{\gamma_1, ..., \gamma_k\}))$, or $\Diamond_s (\psi_{s,1} \vee ... \vee \psi_{s,l_s}) \notin \Pi(\varphi \wedge \neg(\gamma_1 \vee ... \vee \gamma_k \vee (\Box_s (\chi_{s,1} \vee \psi_{s,1} \vee ... \vee \psi_{s,l_s}) \vee ... \vee \Box_s (\chi_{s,m_s} \vee \psi_{s,1} \vee ... \vee \psi_{s,l_s})) \vee \bigvee_{\substack{1 \leq i \leq n \\ i \neq s}} (\Diamond_i \psi_{i,1} \vee ... \vee \Diamond_i \psi_{i,l_i} \vee \Box_i \chi_{i,1} \vee ... \vee \Box_i \chi_{i,m_i}))$ for some s, or $\Box_i (\chi_{i,j} \wedge \neg\psi_{i,1} \wedge ... \wedge \neg\psi_{i,l_i}) \notin \Pi(\varphi \wedge \neg(\lambda \setminus \{\Box\chi_{i,j}\}))$ for some i and j.

Proof. We will only consider the case where $\varphi \models \lambda$ because if $\varphi \not\models \lambda$ then we immediately get the result. Suppose then that $\lambda \notin \Pi(\varphi)$ and $\varphi \models \lambda$. By Definition 3.2.1, there must be some $\lambda' = \gamma'_1 \vee ... \vee \gamma'_o \vee \bigvee_{i=1}^{n} (\Diamond_i \psi'_{i,1} \vee ... \vee \Diamond_i \psi'_{i,p_i} \vee \Box_i \chi'_{i,1} \vee ... \vee \Box_i \chi'_{i,q_i})$ such that $\varphi \models \lambda' \models \lambda \not\models \lambda'$. Since $\lambda \not\models \lambda'$, by Proposition 2.3.3 we know that either $\gamma_1 \vee ... \vee \gamma_k \not\models \gamma'_1 \vee ... \vee \gamma'_o$, or $\psi_{s,1} \vee ... \vee \psi_{s,l_s} \not\models \psi'_{s,1} \vee ... \vee \psi'_{s,p_s}$ for some s, or for some s and t we have $\chi_{s,t} \not\models \chi'_{s,u} \vee \psi'_{s,1} \vee ... \vee \psi'_{s,p_s}$ for all u.

We begin with the case where $\gamma_1 \vee ... \vee \gamma_k \not\models \gamma'_1 \vee ... \vee \gamma'_o$. As $\lambda' \models \lambda$ and $\not\models \lambda$, by Theorem 2.3.3, we know that for every $1 \leq i \leq n$: $\psi'_{i,1} \vee ... \vee \psi'_{i,p_i} \models \psi_{i,1} \vee ... \vee \psi_{i,l_i}$ and for every $1 \leq u \leq q_i$ there is some $1 \leq v \leq m_i$ such that $\chi'_{i,u} \models \psi_{i,1} \vee ... \vee \psi_{i,l_i} \vee \chi_{i,v}$. It follows then (also by Theorem 2.3.3) that $\varphi \models \lambda' \models \gamma'_1 \vee ... \vee \gamma'_o \vee \bigvee_{i=1}^{n} (\Diamond_i \psi_{i,1} \vee ... \vee \Diamond_i \psi_{i,l_i} \vee \Box_i \chi_{i,1} \vee ... \vee \Box_i \chi_{i,m_i})$, and hence that $\varphi \wedge \neg(\lambda \setminus \{\gamma_1, ..., \gamma_k\}) \models \gamma'_1 \vee ... \vee \gamma'_o$. As $\gamma'_1 \vee ... \vee \gamma'_o \models \gamma_1 \vee ... \vee \gamma_k \not\models \gamma'_1 \vee ... \vee \gamma'_o$, we have found an implicate of $\varphi \wedge \neg(\lambda \setminus \{\gamma_1, ..., \gamma_k\})$ which is stronger than $\gamma_1 \vee ... \vee \gamma_k$, so $\gamma_1 \vee ... \vee \gamma_k \notin \Pi(\varphi \wedge \neg(\lambda \setminus \{\gamma_1, ..., \gamma_k\}))$.

Next suppose that $\psi_{s,1} \vee ... \vee \psi_{s,l_s} \not\models \psi'_{s,1} \vee ... \vee \psi'_{s,p_s}$. As $\lambda' \models \lambda$ and $\not\models \lambda$, it follows from Theorem 2.3.3 that $\gamma'_1 \vee ... \vee \gamma'_o \models \gamma_1 \vee ... \vee \gamma_k$ and for every $1 \leq i \leq n$: $\psi'_{i,1} \vee ... \vee \psi'_{i,p_i} \models \psi_{i,1} \vee ... \vee \psi_{i,l_i}$ and for every $1 \leq u \leq q_i$ there is some $1 \leq v \leq m_i$ such that $\chi'_{i,u} \models \psi_{i,1} \vee ... \vee \psi_{i,l_i} \vee \chi_{i,v}$. We thereby obtain

$$\begin{aligned}\varphi \models \lambda' \models\ & \gamma_1 \vee ... \vee \gamma_k \vee (\Diamond_s \psi'_{s,1} \vee ... \vee \Diamond_s \psi'_{s,p_s}) \\ & \vee (\Box_s (\chi_{s,1} \vee \psi_{s,1} \vee ... \vee \psi_{s,l_s}) \vee ... \vee \Box_s (\chi_{s,m_s} \vee \psi_{s,1} \vee ... \vee \psi_{s,l_s})) \\ & \vee \bigvee_{\substack{1 \leq i \leq n \\ i \neq s}} (\Diamond_i \psi_{i,1} \vee ... \vee \Diamond_i \psi_{i,l_i} \vee \Box_i \chi_{i,1} \vee ... \vee \Box_i \chi_{i,m_i})\end{aligned}$$

From this, we can infer that

$$\begin{aligned}\varphi \wedge \neg(\gamma_1 \vee ... \vee \gamma_k \vee (\Box_s (\chi_{s,1} \vee \psi_{s,1} \vee ... \vee \psi_{s,l_s}) \vee ... \vee \Box_s (\chi_{s,m_s} \vee \psi_{s,1} \vee ... \vee \psi_{s,l_s})) \\ \vee \bigvee_{\substack{1 \leq i \leq n \\ i \neq s}} (\Diamond_i \psi_{i,1} \vee ... \vee \Diamond_i \psi_{i,l_i} \vee \Box_i \chi_{i,1} \vee ... \vee \Box_i \chi_{i,m_i})) \\ \models \Diamond_s \psi'_{s,1} \vee ... \vee \Diamond \psi'_{s,p_s} \models \Diamond_s \psi_{s,1} \vee ... \vee \Diamond \psi_{s,l_s} \not\models \Diamond_s \psi'_{s,1} \vee ... \vee \Diamond \psi'_{s,p_s}\end{aligned}$$

As $\Diamond_s \psi_{s,1} \vee ... \vee \Diamond_s \psi_{s,l_s} \equiv \Diamond_s (\psi_{s,1} \vee ... \vee \psi_{s,l_s})$, it follows that $\Diamond_s (\psi_{s,1} \vee ... \vee \psi_{s,l_s}) \notin \Pi(\varphi \wedge \neg(\gamma_1 \vee ... \vee \gamma_k \vee (\Box_s (\chi_{s,1} \vee \psi_{s,1} \vee ... \vee \psi_{s,l_s}) \vee ... \vee \Box_s (\chi_{s,m_s} \vee \psi_{s,1} \vee ... \vee \psi_{s,l_s})) \vee \bigvee_{\substack{1 \leq i \leq n \\ i \neq s}} (\Diamond_i \psi_{i,1} \vee ... \vee \Diamond_i \psi_{i,l_i} \vee \Box_i \chi_{i,1} \vee ... \vee \Box_i \chi_{i,m_i}))$.

Finally suppose that $\chi_{s,t} \not\models \chi'_{s,u} \vee \psi'_{s,1} \vee ... \vee \psi'_{s,p_s}$ for all u and furthermore that $\psi_{i,1} \vee ... \vee \psi_{i,l_i} \models \psi'_{i,1} \vee ... \vee \psi'_{i,p_i}$ for all $1 \leq i \leq n$ (we have already shown the result holds whenever $\psi_{i,1} \vee ... \vee \psi_{i,l_i} \not\models \psi'_{i,1} \vee ... \vee \psi'_{i,p_i}$ for some i). Now $\Box_s (\chi_{s,t} \wedge \neg\psi_{s,1} \wedge ... \wedge \neg\psi_{s,l_s})$ is an implicate of $\varphi \wedge \neg(\lambda \setminus \{\Box_s \chi_{s,t}\}))$ so to show that $\Box_s (\chi_{s,t} \wedge \neg\psi_{s,1} \wedge ... \wedge \neg\psi_{s,l_s})$ is not a prime implicate of $\varphi \wedge \neg(\lambda \setminus \{\Box_s \chi_{s,t}\}))$, we must find some stronger implicate. Set $S = \{v \in \{1, ..., q_s\} : \chi'_{s,v} \models \chi_{s,t} \vee \psi_{s,1} \vee ... \vee \psi_{s,l_s}$ and $\chi'_{s,v} \not\models \chi_{s,w} \vee \psi_{s,1} \vee ... \vee \psi_{s,l_s}$ for $w \neq t\}$. We note that there must be at least one element in S as we have assumed $\varphi \not\models \lambda \setminus \{\Box_s \chi_{s,t}\}$. We also know that:

- $\gamma'_1 \vee ... \vee \gamma'_o \models \gamma_1 \vee ... \vee \gamma_k$
- for every $1 \leq i \leq n$: $\psi'_{i,1} \vee ... \vee \psi'_{i,p_i} \models \psi_{i,1} \vee ... \vee \psi_{i,l_i}$
- for every $i \neq s$ and $1 \leq v \leq q_i$: there is some w such that $\chi'_{i,v} \models \chi_{i,w} \vee \psi_{i,1} \vee ... \vee \psi_{i,l_i}$
- for every $1 \leq v \leq q_s$ with $v \notin S$: there is some $w \neq t$ such that $\chi'_{s,v} \models \chi_{s,w} \vee \psi_{s,1} \vee ... \vee \psi_{s,l_s}$
- for every $v \in S$: $\chi'_{s,v} \models \chi'_{s,v}$

From these statements, we get that

$$\varphi \models \lambda' \models \quad \gamma_1 \vee ... \vee \gamma_k \vee \bigvee_{i=1}^{n} (\Diamond_i \psi_{i,1} \vee ... \vee \Diamond_i \psi_{i,l_i}) \vee \bigvee_{\substack{1 \leq i \leq n \\ i \neq s}} (\Box_i \chi_{i,1} \vee ... \vee \Box_i \chi_{i,m_i}) \vee \bigvee_{\substack{1 \leq w \leq m_s \\ w \neq t}} \Box_s \chi_{s,w} \vee \bigvee_{v \in S} \Box_s \chi'_{s,v}$$

It follows that $\varphi \wedge \neg(\lambda \setminus \{\Box_s \chi_{s,t}\}) \models \bigvee_{v \in S} \Box_s (\chi'_{s,v} \wedge \neg\psi_{s,1} \wedge ... \wedge \psi_{s,l_s})$, which means that $\bigvee_{v \in S} \Box_s (\chi'_{s,v} \wedge \neg\psi_{s,1} \wedge ... \wedge \psi_{s,l_s})$ is an implicate of $\varphi \wedge \neg(\lambda \setminus \{\Box_s \chi_{s,t}\})$. Moreover, $\bigvee_{v \in S} \Box_s (\chi'_{s,v} \wedge \neg\psi_{s,1} \wedge ... \wedge \psi_{s,l_s}) \models \Box_s (\chi_{s,t} \wedge \neg\psi_{s,1} \wedge ... \wedge \neg\psi_{s,l_s})$ since by construction $\chi'_{s,v} \models \chi_{s,t} \vee \psi_{s,1} \vee ... \vee \psi_{s,l_s}$ for every $v \in S$.

It remains to be shown that $\Box_s (\chi_{s,t} \wedge \neg\psi_{s,1} \wedge ... \wedge \neg\psi_{s,l_s}) \not\models \bigvee_{v \in S} \Box_s (\chi'_{s,v} \wedge \neg\psi_{s,1} \wedge ... \wedge \psi_{s,l_s})$. Suppose for a contradiction that the contrary holds. Then $\Box_s (\chi_{s,t} \wedge \neg\psi_{s,1} \wedge ... \wedge \neg\psi_{s,l_s}) \models \bigvee_{v \in S} \Box_s (\chi'_{s,v} \wedge \neg\psi_{s,1} \wedge ... \wedge \psi_{s,l_s})$, so by Theorem 2.3.1, there must be some $v \in S$ for which $\chi_{s,t} \wedge \neg\psi_{s,1} \wedge ... \wedge \neg\psi_{s,l_s} \models \chi'_{s,v} \wedge \neg\psi_{s,1} \wedge ... \wedge \neg\psi_{s,l_s}$. But then $\chi_{s,t} \models \chi'_{s,v} \vee \psi_{s,1} \vee ... \vee \psi_{s,l_s}$, and thus $\chi_{s,t} \models \chi'_{s,v} \vee \psi'_{s,1} \vee ... \vee \psi'_{s,p_s}$ since we have assumed $\psi_{s,1} \vee ... \vee \psi_{s,l_s} \models \psi'_{s,1} \vee ... \vee \psi'_{s,p_s}$. This contradicts our earlier assumption that $\chi_{s,t} \not\models \chi'_{s,u} \vee \psi'_{s,1} \vee ... \vee \psi'_{s,p_s}$ for all u. Thus, we have shown that $\Box_s (\chi_{s,t} \wedge \neg\psi_{s,1} \wedge ... \wedge \neg\psi_{s,l_s}) \not\models \bigvee_{v \in S} \Box_s (\chi'_{s,v} \wedge \neg\psi_{s,1} \wedge ... \wedge \psi_{s,l_s})$, which means $\Box_s (\chi_{s,t} \wedge \neg\psi_{s,1} \wedge ... \wedge \neg\psi_{s,l_s}) \notin \Pi(\varphi \wedge \neg(\lambda \setminus \{\Box_s \chi_{s,t}\}))$. □

Proof of Theorem 4.2.3. The forward direction was shown in Lemma 4.2.4. The other direction follows from Lemma 4.2.5 together with the hypothesis that $\chi_{i,j} \equiv \chi_{i,j} \vee \psi_{i,1} \vee ... \vee \psi_{i,l_i}$ for all $1 \leq i \leq n$ and $1 \leq j \leq m_i$ (which ensures that $\varphi \wedge \neg(\gamma_1 \vee ... \vee \gamma_k \vee (\Box_s (\chi_{s,1} \vee \psi_{s,1} \vee ... \vee \psi_{s,l_s}) \vee ... \vee \Box_s (\chi_{s,m_s} \vee \psi_{s,1} \vee ... \vee \psi_{s,l_s})) \vee \bigvee_{\substack{1 \leq i \leq n \\ i \neq s}} (\Diamond_i \psi_{i,1} \vee ... \vee \Diamond_i \psi_{i,l_i} \vee \Box_i \chi_{i,1} \vee ... \vee \Box_i \chi_{i,m_i}) \equiv \varphi \wedge \neg(\lambda \setminus \{\Diamond_s \psi_{s,1}, ..., \Diamond_s \psi_{s,l_s}\}))$. □

We remark that the restriction of Theorem 4.2.3 to clauses for which $\chi_{i,j} \equiv \chi_{i,j} \vee \psi_{i,1} \vee ... \vee \psi_{i,l_i}$ for all i and j and for which $\lambda \not\equiv \lambda \setminus \{l\}$ for all disjuncts l is required. If we drop the former restriction, then there are some non-prime implicates that satisfy all three conditions, as the following example demonstrates.

Example 4.2.6.
Consider the formula $\varphi = \Diamond(a \wedge b \wedge c) \vee \Box a$ and the clause $\lambda = \Diamond(a \wedge b) \vee \Box(a \wedge \neg b)$. It can be easily shown that λ is an implicate of φ, but λ is not a prime implicate of φ since there exist stronger implicates (e.g. φ itself). Nonetheless, it can be verified that both $\Box(a \wedge \neg b \wedge \neg(a \wedge b)) \in \Pi(\varphi \wedge \neg(\lambda \setminus \{\Box(a \wedge \neg b)\}))$ and $\Diamond(a \wedge b) \in \Pi(\varphi \wedge \neg(\lambda \setminus \{\Diamond(a \wedge b)\}))$.

If we drop the requirement that $\lambda \not\equiv \lambda \setminus \{l\}$ for all disjuncts l, then there are prime implicates which fail to satisfy one of the three conditions, as witnessed by the following example.

Example 4.2.7.
Consider the formula $\Box a$ and the clause $\Box a \vee \Box(a \wedge b)$. We have $\Box(a \wedge b) \notin \Pi(\Box a \wedge \neg(\Box a))$ even though $\Box a \vee \Box(a \wedge b)$ is a prime implicate of $\Box a$.

These two restrictions are without loss of generality however since every clause can be transformed into an equivalent clause satisfying them. For the first restriction, we replace each $\Box_i \chi_{i,j}$ by $\Box_i (\chi_{i,h} \vee \psi_{i,1} \vee ... \vee \psi_{i,l_i})$, thereby transforming a clause $\gamma_1 \vee ... \vee \gamma_k \vee \bigvee_{i=1}^{n}(\Diamond_i \psi_{i,1} \vee ... \vee \Diamond_i \psi_{i,l_i} \vee \Box_i \chi_{i,1} \vee ... \vee \Box_i \chi_{i,m_i})$ into the equivalent $\gamma_1 \vee ... \vee \gamma_k \vee \bigvee_{i=1}^{n}(\Diamond_i \psi_{i,1} \vee ... \vee \Diamond_i \psi_{i,l_i} \vee \Box_i (\chi_{i,1} \vee \psi_{i,1} \vee ... \vee \psi_{i,l_i}) \vee ... \vee \Box_i (\chi_{i,m_i} \vee \psi_{i,1} \vee ... \vee \psi_{i,l_i}))$. Then to make the clause satisfy the second restriction, we simply remove from λ those disjuncts l satisfying $\lambda \equiv \lambda \setminus \{l\}$ until no such disjuncts remains.

Theorem 4.2.3 shows us how prime implicate recognition can be split into three more specialized sub-tasks, but it does not tell us how to carry out these tasks. Thus, in order to turn this theorem into an algorithm for prime implicate recognition, we need to figure out how to test whether a propositional clause, a $\Box$-literal, or a $\Diamond$-literal is a prime implicate of a formula.

4.2.4 Prime implicate recognition for propositional clauses

Determining whether a propositional clause is a prime implicate of a formula in $\mathcal{K}_n$ is conceptually no more difficult than determining whether a propositional clause is a prime implicate of a propositional formula. We first ensure that the clause is an implicate of the formula and then make sure that all literals appearing in the clause are necessary.

Theorem 4.2.8.
Let φ be a formula of $\mathcal{K}_n$, and let γ be a non-tautologous propositional clause such that $\varphi \models \gamma$ and such that there is no literal l in γ such that $\gamma \equiv \gamma \setminus \{l\}$. Then $\gamma \in \Pi(\varphi)$ if and only if $\varphi \not\models \gamma \setminus \{l\}$ for all l in γ.

Proof. Consider a formula φ and a non-tautologous propositional clause λ such that $\varphi \models \lambda$ and such that there is no literal l in λ such that $\lambda \equiv \lambda \setminus \{l\}$. Suppose that $\varphi \models \lambda \setminus \{l\}$ for some l in λ. As we know that $\lambda \not\equiv \lambda \setminus \{l\}$, it follows that $\lambda \setminus \{l\}$ is an implicate of φ which is strictly stronger than λ, so λ is not a prime implicate

of φ. For the other direction, suppose that $\lambda \notin \Pi(\varphi)$. Then it must be the case that there is some clause ρ such that $\varphi \models \rho \models \lambda \not\models \rho$. Since $\rho \models \lambda$, it follows from Theorem 2.3.2 that each literal in ρ is a propositional literal of λ or is inconsistent. If all of the literals in ρ are inconsistent, then both ρ and φ must be inconsistent, so clearly $\varphi \models \gamma \setminus \{l\}$ for every l in γ. Otherwise, ρ is equivalent to a propositional clause, and more specifically to a propositional clause containing only those literals appearing in λ (since $\rho \models \lambda$). As ρ is strictly stronger than λ, there must be some literal l in λ which does not appear in ρ. But that means $\rho \models \lambda \setminus \{l\}$ and so $\varphi \models \lambda \setminus \{l\}$, completing the proof. □

4.2.5 Prime implicate recognition for □-formulae

We now move on to the problem of deciding whether a clause of the form $\Box_i \chi$ is a prime implicate of a formula φ. We remark that if $\Box_i \chi$ is implied by φ, then it must also be implied by each of the terms T in $\mathcal{T}$=**Dnf**(φ). But if $T \models \Box_i \chi$ and T is satisfiable, then it must be the case that the conjunction of the $\Box_i$-literals in T implies $\Box_i \chi$. This means that the formula $\bigvee_{T \in \mathcal{T}; T \not\models \bot} \Box_i \bigwedge_{\zeta \in Box_i(T)} \zeta$ is an implicate of φ which implies $\Box_i \chi$, and moreover it is the strongest such implicate. It follows then that $\Box_i \chi$ is a prime implicate of φ just in the case that $\Box_i \chi$ entails this formula. But we know from results in Chapter 2 that the latter holds if and only if $\chi \models \bigwedge_{\zeta \in Box_i(T)} \zeta$ for some satisfiable T. Thus, by comparing the formula χ with the formulae $\bigwedge_{\zeta \in Box_i(T)} \zeta$ associated with the satisfiable terms of φ, we can decide whether or not $\Box_i \chi$ is a prime implicate of φ.

Theorem 4.2.9.
*Let φ be a formula of $\mathcal{K}_n$, and let $\lambda = \Box_i \chi$ be a non-tautologous clause such that $\varphi \models \lambda$. Then $\lambda \in \Pi(\varphi)$ if and only if there exists some satisfiable term $T \in$***Dnf***(φ) such that $\chi \models \bigwedge_{\zeta \in Box_i(T)} \zeta$.*

Proof. Let φ be some formula, and let $\lambda = \Box_i \chi$ be a non-tautologous clause such that $\varphi \models \lambda$. Set $\mathcal{T}$ =**Dnf**(φ). For the first direction, suppose that there is no satisfiable term $T \in \mathcal{T}$ such that $\chi \models \bigwedge_{\zeta \in Box_i(T)} \zeta$. We first remark that since $\varphi \models \lambda$, every T must entail λ. Moreover, since λ is a non-tautologous $\Box_i$-formula, every term $T \in \mathcal{T}$ must be either unsatisfiable or such that $Box_i(T) \neq \emptyset$ and $\bigwedge_{\zeta \in Box_i(T)} \zeta \models \chi$. If $\mathcal{T}$ contains only unsatisfiable terms, then $\lambda \models \bot$ but $\Box_i \chi \not\models \bot$, so λ is not a prime implicate. If there is at least one satisfiable term in $\mathcal{T}$, then consider the clause

$$\lambda' = \bigvee_{T \in \mathcal{T} : T \not\models \bot} \Box_i \bigwedge_{\zeta \in Box_i(T)} \zeta$$

For every T, we must have $T \models \lambda'$ since either T is unsatisfiable or it entails $\Box_i \bigwedge_{\zeta \in Box_i(T)} \zeta$. This means that $\varphi \models \lambda'$. We also have $\lambda' \models \lambda$, because $\bigwedge_{\zeta \in Box_i(T)} \zeta \models \chi$ for every satisfiable T. Finally, we must have $\lambda \not\models \lambda'$, because of statement 8 of Theorem 2.3.1 and our earlier assumption that $\chi \not\models \bigwedge_{\zeta \in Box_i(T)} \zeta$ for every satisfiable $T \in \mathcal{T}$. So we have $\varphi \models \lambda' \models \lambda \not\models \lambda'$, which means that λ is not a prime implicate of φ.

For the other direction, suppose that $\Box_i \chi$ is not a prime implicate of φ. If $\mathcal{T}$ is composed entirely of unsatisfiable terms, then there clearly cannot exist any satisfiable term $T \in \mathcal{T}$ such that $\chi \models \bigwedge_{\zeta \in Box_i(T)} \zeta$. Suppose then that $\mathcal{T}$ contains at least one satisfiable term. We intend to show that the clause

$$\lambda' = \bigvee_{T \in \mathcal{T} : T \not\models \bot} \Box_i \bigwedge_{\zeta \in Box_i(T)} \zeta$$

is a prime implicate of T. To do so, we let κ be some implicate of φ which implies λ'. Now since λ' is a non-tautologous disjunction of $\Box$-formulae, it follows from Theorem 2.3.2 that $\kappa \equiv \Box_i \gamma_1 \vee ... \vee \Box_i \gamma_m$ for some formulae γ_j. As $\varphi \models \kappa$, we must have $T \models \Box_i \gamma_1 \vee ... \vee \Box_i \gamma_m$ for all $T \in \mathbf{Dnf}(\varphi)$. But that can only be the case if $\Box_i \bigwedge_{\zeta \in Box_i(T)} \zeta \models \Box_i \gamma_1 \vee ... \vee \Box_i \gamma_m$ for all satisfiable T (Theorem 2.3.1), which means $\lambda' \models \Box_i \gamma_1 \vee ... \vee \Box_i \gamma_m$. As λ' implies every implicate of φ that implies it, it must be a prime implicate of φ. But this means that $\Box_i \chi \not\models \lambda'$, since we have assumed that $\Box_i \chi$ is not a prime implicate of φ. It follows from statement 8 of Theorem 2.3.1 that $\chi \not\models \Box_i \bigwedge_{\zeta \in Box_i(T)} \zeta$ for all satisfiable $T \in \mathbf{Dnf}(\varphi)$. □

4.2.6 Prime implicate recognition for ◇-formulae

Finally let us turn to the problem of deciding whether a clause $\Diamond\psi$ is a prime implicate of a formula φ. Now we know by **Covering** that if $\Diamond\psi$ is an implicate of φ, then there must be some prime implicate π of φ which implies $\Diamond\psi$. It follows from Theorem 2.3.2 that π must be a disjunction of $\Diamond$-literals, and from Theorem 4.1.4 that π is equivalent to a disjunction $\bigvee_{T \in \mathbf{Dnf\text{-}4}(\varphi)} \Diamond d_T$ where $\Diamond d_T$ is an element of $\Delta(T)$ for every T (refer back to Figure 4.1 for the definition of $\Delta(T)$). According to Definition 3.2.1, $\Diamond\psi$ is a prime implicate of φ just in the case that $\Diamond\psi \models \bigvee_{T \in \mathbf{Dnf\text{-}4}(\varphi)} \Diamond d_T$, or equivalently $\psi \models \bigvee_{T \in \mathbf{Dnf\text{-}4}(\varphi)} d_T$. Thus, $\Diamond\psi$ is not a prime implicate of φ just in the case that there is a choice of $\Diamond d_T \in \Delta(T)$ for each $T \in \mathbf{Dnf\text{-}4}(\varphi)$ such that $\bigvee_{T \in \mathbf{Dnf\text{-}4}(\varphi)} d_T \models \psi$ and $\psi \not\models \bigvee_{T \in \mathbf{Dnf\text{-}4}(\varphi)} d_T$.

Testing directly whether ψ entails some formula $\bigvee_{T \in \mathbf{Dnf\text{-}4}(\varphi)} d_T$ could take exponential space in the worst case since there may be exponentially many terms in $\mathbf{Dnf\text{-}4}(\varphi)$. Luckily, however, we can get around this problem by exploiting the structure of the formula $\bigvee_{T \in \mathbf{Dnf\text{-}4}(\varphi)} d_T$. We remark that because of the way $\Delta(T)$

is defined the formula d_T must be a conjunction of formulae ζ such that $\Box\zeta$ or $\Diamond\zeta$ appears in $\mathbf{Nnf}(\varphi)$ outside the scope of modal operators – we will use $\mathcal{X}$ to denote the set of formulae ζ satisfying this condition. We show in what follows that $\psi \not\models \bigvee_{T\in\mathbf{Dnf\text{-}4}(\varphi)} d_T$ implies the existence of a subset $S \subseteq \mathcal{X}$ such that (a) $\psi \not\models \bigvee_{\sigma\in S} \sigma$ and (b) every d_T has at least one conjunct from the set S. Conversely, the existence of such a subset of $\mathcal{X}$ implies $\psi \not\models \bigvee_{T\in\mathbf{Dnf\text{-}4}(\varphi)} d_T$. This observation is the basis for the algorithm **Test$\Diamond$PI** given in Figure 4.2. The basic idea behind the algorithm **Test$\Diamond$PI** is to try out each of the different subsets of $\mathcal{X}$ in order to see whether some subset satisfies the aforementioned conditions. If we find a suitable subset, this proves that $\Diamond\psi$ is not a prime implicate, and if no such subset exists, then we can be sure there is no stronger implicate than $\Diamond\psi$. The algorithm can be shown to run in polynomial space since there can be at most $|\varphi|$ elements in $\mathcal{X}$, and we can consider the terms in $\mathbf{Dnf}(\varphi)$ one at a time.

Algorithm 4.2 Test$\Diamond$PI

Input: a clause $\Diamond_i\psi$ and a formula φ such that $\varphi \models \Diamond_i\psi$

Output: **yes** if $\Diamond_i\psi$ is a prime implicate of φ, otherwise **no**

(1) If $\mathbf{Sat}(\varphi)$=**no**, return **yes** if $\mathbf{Sat}(\psi)$=**no**, else return **no**.

(2) Set $\mathcal{X}$ equal to the set of formulae ζ such that $\Box_i\zeta$ or $\Diamond_i\zeta$ appears in $\mathbf{Nnf}(\varphi)$ outside the scope of modal operators.

(3) For each $S \subseteq \mathcal{X}$, test whether the following two conditions hold:

(a) $\mathbf{Entails}(\psi, \bigvee_{\sigma\in S}\sigma)$=**no**

(b) for each $T_j \in \mathbf{Dnf}(\varphi)$ such that $\mathbf{Sat}(T_j)$=**yes**, there exists conjuncts $\Diamond\eta_j, \Box\mu_{j,1}, ..., \Box\mu_{j,k_j}$ of T_j such that:

(i) $\{\eta_j, \mu_{j,1}, ..., \mu_{j,k_j}\} \cap S \neq \emptyset$

(ii) $\mathbf{Entails}(\Diamond_i(\eta_j \wedge \mu_{j,1} \wedge ... \wedge \mu_{j,k_j}), \Diamond_i\psi)$=**yes**

Return **no** if some S satisfies these conditions, and **yes** otherwise.

We will split the proof of correctness of **Test$\Diamond$PI** into two parts.

Lemma 4.2.10.
If $\Diamond_i\psi$ is not a prime implicate of φ, the algorithm **Test$\Diamond$PI** *returns* ***no*** *on input $(\Diamond_i\psi, \varphi)$.*

Proof. Suppose that $\Diamond_i\psi$ is not a prime implicate of φ. If φ is unsatisfiable, then $\Diamond_i\psi$ must be satisfiable, so we will have $\mathbf{Sat}(\varphi)$=**no** and $\mathbf{Sat}(\psi)$=**yes**, and the algorithm will return **no** in Step 1. Otherwise, as we have assumed that the input $\Diamond_i\psi$ is an implicate of φ, there must be some clause λ such that $\varphi \models \lambda \models \Diamond_i\psi$ but $\Diamond_i\psi \not\models \lambda$. As $\lambda \models \Diamond_i\psi$, it follows from Theorem 2.3.2 that λ is equivalent to a disjunction of $\Diamond_i$-formulae, and hence to some clause $\Diamond_i\psi'$.

We know from Theorem 2.4.4 that φ is equivalent to the disjunction of terms in $\mathbf{Dnf}(\varphi)$. It must thus be the case that $T_j \models \Diamond_i \psi'$ for all $T_j \in \mathbf{Dnf}(\varphi)$. This means that for every satisfiable T_j, there exists a set $\{\Diamond_i \eta_j, \Box_i \mu_{j,1}, ..., \Box_i \mu_{j,k_j}\}$ of conjuncts of T_j such that $\Diamond_i (\eta_j \wedge \mu_{j,1} \wedge ... \wedge \mu_{j,k_j}) \models \Diamond_i \psi'$, otherwise T_j would fail to imply $\Diamond_i \psi'$. Moreover, all of the elements of $\{\Diamond_i \eta_j, \Box_i \mu_{j,1}, ..., \Box_i \mu_{j,k_j}\}$ must appear in the NNF of φ outside modal operators, so the formulae $\eta_j, \mu_{j,1}, ..., \mu_{j,k_j}$ must all be elements of the set $\mathcal{X}$. It is immediate that both

$$\Diamond_i \bigvee_{j:T_j \not\models \bot} (\eta_j \wedge \mu_{j,1} \wedge ... \wedge \mu_{j,k_j}) \models \Diamond_i \psi' \models \Diamond_i \psi \tag{4.3}$$

and

$$\Diamond_i \psi \not\models \Diamond_i \bigvee_{j:T_j \not\models \bot} (\eta_j \wedge \mu_{j,1} \wedge ... \wedge \mu_{j,k_j})$$

The latter implies that the formula $\Diamond_i \psi \wedge \neg(\Diamond_i \bigvee_{j:T_j \not\models \bot} (\eta_j \wedge \mu_{j,1} \wedge ... \wedge \mu_{j,k_j}))$ must be consistent, which means that

$$\psi \wedge \neg(\bigvee_{j:T_j \not\models \bot} (\eta_j \wedge \mu_{j,1} \wedge ... \wedge \mu_{j,k_j}) \equiv \psi \wedge \bigwedge_{j:T_j \not\models \bot} (\neg\eta_j \vee \neg\mu_{j,1} \vee ... \vee \neg\mu_{j,k_j})$$

must be consistent as well. But then it must be the case that we can select for each j with $T_j \not\models \bot$ some $\sigma_j \in \{\eta_j, \mu_{j,1}, ..., \mu_{j,k_j}\}$ such that $\psi \wedge \bigwedge_{j:T_j \not\models \bot} \neg\sigma_j$ is consistent. Let S be the set of σ_j. The set S satisfies the condition of the algorithm since:

- $S \subseteq \mathcal{X}$
- $\psi \not\models \bigvee_{\sigma \in S} \sigma$ (because we know $\psi \wedge \bigwedge_{j:T_j \not\models \bot} \neg\sigma_j$ to be consistent), and hence $\mathbf{Entails}(\psi, \bigvee_{\sigma \in S} \sigma)$=**no**
- for each satisfiable $T_j \in \mathbf{Dnf}(\varphi)$, we have found a set $\{\Diamond\eta_j, \Box\mu_{j,1}, ..., \Box\mu_{j,k_j}\}$ of conjuncts of T_j such that:
  - $\{\eta_j, \mu_{j,1}, ..., \mu_{j,k_j}\} \cap S \neq \emptyset$ (since S contains $\sigma_j \in \{\eta_j, \mu_{j,1}, ..., \mu_{j,k_j}\}$)
  - $\Diamond_i (\eta_j \wedge \mu_{j,1} \wedge ... \wedge \mu_{j,k_j}) \models \Diamond_i \psi$ (follows from (4.3) above), and hence $\mathbf{Entails}(\Diamond_i (\eta_j \wedge \mu_{j,1} \wedge ... \wedge \mu_{j,k_j}), \Diamond_i \psi)$=**yes**

Since there exists a set $S \subseteq \mathcal{X}$ satisfying these conditions, the algorithm **Test$\Diamond$PI** returns **no**. □

Lemma 4.2.11.
If the algorithm **Test$\Diamond$PI** *returns* ***no*** *on input* $(\Diamond_i \psi, \varphi)$*, then* $\Diamond_i \psi$ *is not a prime implicate of* φ*.*

Proof. There are two cases in which the algorithm returns **no**: either $\mathbf{Sat}(\varphi)$=**no** and $\mathbf{Sat}(\psi)$=**yes**, or there is some $S \subseteq \mathcal{X}$ which satisfies both conditions (a) and (b). In the first case, $\Diamond_i \psi$ is clearly not a prime implicate since $\varphi \models \bot$ and $\Diamond_i \psi \not\models \bot$

(by Theorem 2.5.8). We now examine the second case in more detail.

Suppose that $S \subseteq \mathcal{X}$ is such that:

(a) $\psi \not\models \bigvee_{\sigma \in S} \sigma$

(b) for each satisfiable $T_j \in \mathbf{Dnf}(\varphi)$, there exists a set of conjuncts $\{\Diamond\eta_j, \Box\mu_{j,1}, ..., \Box\mu_{j,k_j}\}$ of T_j such that:

(i) $\{\eta_j, \mu_{j,1}, ..., \mu_{j,k_j}\} \cap S \neq \emptyset$

(ii) $\Diamond_i (\eta_j \wedge \mu_{j,1} \wedge ... \wedge \mu_{j,k_j}) \models \Diamond_i \psi$

Let α be the clause $\bigvee_{j:T_j \not\models \bot} \Diamond(\eta_j \wedge \mu_{j,1} \wedge ... \wedge \mu_{j,k_j})$. We remark that for each satisfiable T_j, we have $T_j \models \Diamond_i (\eta_j \wedge \mu_{j,1} \wedge ... \wedge \mu_{j,k_j})$, and hence $\bigvee_j T_j \models \alpha$. Because of Corollary 2.4.5, we also have $\varphi \equiv \bigvee_j T_j$, which gives us $\varphi \models \alpha$. From 2 (b) (ii), we have that $\Diamond_i (\eta_j \wedge \mu_{j,1} \wedge ... \wedge \mu_{j,k_j}) \models \Diamond_i \psi$ for every i, and hence $\bigvee_{j:T_j \not\models \bot} \Diamond_i (\eta_j \wedge \mu_{j,1} \wedge ... \wedge \mu_{j,k_j}) \models \Diamond_i \psi$ which yields $\alpha \models \Diamond_i \psi$. From 2 (b) (i), we have that $\{\eta_j, \mu_{j,1}, ..., \mu_{j,k_j}\} \cap S \neq \emptyset$ and hence that for every j satisfying $T_j \not\models \bot$, there is some $\sigma \in S$ such that $\eta_j \wedge \mu_{j,1} \wedge ... \wedge \mu_{j,k_j} \models \sigma$. From this we can infer that $\bigvee_{j:T_j \not\models \bot} \Diamond_i (\eta_j \wedge \mu_{j,1} \wedge ... \wedge \mu_{j,k_j}) \models \bigvee_{\sigma \in S} \Diamond_i \sigma$, and hence $\alpha \models \Diamond_i \bigvee_{\sigma \in S} \sigma$. But we know from 2 (a) and Theorem 2.3.1 that $\Diamond_i \psi \not\models \Diamond_i \bigvee_{\sigma \in S} \sigma$. It follows then that $\Diamond_i \psi \not\models \alpha$. Putting all this together, we find that there exists a clause α such that $\varphi \models \alpha \models \Diamond_i \psi$ but $\Diamond_i \psi \not\models \alpha$, and hence that $\Diamond_i \psi$ is not a prime implicate of φ. □

Theorem 4.2.12.
Let φ be a formula, and let $\Diamond_i \psi$ be an implicate of φ. Then the algorithm **Test$\Diamond$PI** *returns* ***yes*** *on input ($\Diamond_i \psi$, φ) if and only if $\Diamond_i \psi$ is a prime implicate of φ.*

Proof. It is clear that **Test$\Diamond$PI** terminates since satisfiability testing always terminates, and there are only finitely many elements in S and terms in $\mathbf{Dnf}(\varphi)$. Lemmas 4.2.10 and 4.2.11 show us that the algorithm always gives the correct response. □

Theorem 4.2.13.
The algorithm **Test$\Diamond$PI** *runs in polynomial space.*

Proof. The algorithm **Sat** runs in polynomial space in its input (Theorem 2.5.6), so Step 1 requires only polynomial space in the length of $|\varphi|$ and $|\psi|$. We next remark that the length of the conjunction of elements in $\mathcal{X}$ is bounded above by the length of the formula $\textsc{Nnf}(\varphi)$, and hence by Theorem 2.4.2 the conjunction of the elements of any particular $S \subseteq \mathcal{X}$ cannot exceed $2|\varphi|$. It follows that running **Entails** on the pair (ψ, $\bigvee_{\sigma \in S} \sigma$) takes only polynomial space in the length of φ and ψ.

Now let us turn to Step 3 (b). We know from Theorem 2.4.7 that **Dnf** runs using only polynomial space, and the length of any T_j in $\mathbf{Dnf}(\varphi)$ can be at most

$2|\varphi|$ (Theorem 2.4.6). It follows that checking whether $\{\eta_j, \mu_{j,1}, ..., \mu_{j,k_j}\} \cap S \neq \emptyset$, or whether **Entails**$(\Diamond_i (\eta_j \wedge \mu_{j,1} \wedge ... \wedge \mu_{j,k_j}), \Diamond_i \psi)$=**yes** can both be accomplished in polynomial space in the length of φ and ψ. We conclude that the algorithm **Test**$\Diamond$**PI** runs in polynomial space.

We now present two examples which illustrate the functioning of **Test**$\Diamond$**PI**.

Example 4.2.14.
We use **Test**$\Diamond$**PI** to test whether the clause $\lambda = \Diamond(a \wedge b)$ is a prime implicate of $\varphi = a \wedge (\Box(b \wedge c) \vee \Box(e \vee f)) \wedge \Diamond(a \wedge b)$.

Step 1: As φ is satisfiable, the algorithm **Sat** returns **yes** on φ, so we pass directly to Step 2.

Step 2: We set $\mathcal{X}$ equal to the set of formulae ζ such that $\Box\zeta$ or $\Diamond\zeta$ appears in **Nnf**(φ) outside the scope of modal operators. In our case, we set $\mathcal{X} = \{b \wedge c, e \vee f, a \wedge b\}$ since $\varphi =$**Nnf**(φ) and $b \wedge c$, $e \vee f$, and $a \wedge b$ are the only formulae satisfying the requirements.

Step 3: We examine each of the different subsets of $\mathcal{X}$ to determine whether they satisfy conditions (a) and (b). In particular, we consider the subset $S = \{b \wedge c, e \vee f\}$. We remark that this subset satisfies condition (a) since $a \wedge b \not\models (b \wedge c) \vee (e \vee f)$ (hence **Entails**$(a \wedge b, (b \wedge c) \vee (e \vee f)\ )$=**no**). In order to check condition (b), we call the procedure **Dnf** on input **Nnf**(φ).The first term output is $T_1 = a \wedge \Box(b \wedge c) \wedge \Diamond(a \wedge b)$. We notice that the conjuncts $\Diamond(a \wedge b)$ and $\Box(b \wedge c)$ of T_1 satisfy conditions (i) and (ii) since $b \wedge c \in S$ and $\Diamond(a \wedge b \wedge (b \wedge c)) \models \lambda$ (hence **Entails**$(\Diamond(a \wedge b \wedge (b \wedge c)), \lambda)$=**yes**). The next and final term output by **Dnf** is $T_2 = a \wedge \Box(e \vee f) \wedge \Diamond(a \wedge b)$. We notice that the conjuncts $\Diamond(a \wedge b)$ and $\Box(e \vee f)$ of T_2 also satisfy conditions (i) and (ii) since $e \vee f \in S$ and $\Diamond(a \wedge b \wedge (e \vee f)) \models \lambda$ (so **Entails**$(\Diamond(a \wedge b \wedge (e \vee f)), \lambda)$=**yes**). That means that we have found a subset S of $\mathcal{X}$ which satisfies conditions (a) and (b), so the algorithm returns **no**. This is the correct output since $\Diamond(a \wedge b \wedge ((b \wedge c) \vee (e \vee f)))$ is an implicate of φ which is strictly stronger than the clause λ.

Example 4.2.15.
We use **Test**$\Diamond$**PI** to test whether the clause $\lambda = \Diamond(a \wedge b \wedge c)$ is a prime implicate of $\varphi = a \wedge (\Box(b \wedge c) \vee \Box(e \vee f)) \wedge \Diamond(a \wedge b) \wedge \neg\Box(e \vee f \vee (a \wedge b \wedge c))$.

Step 1: We proceed directly to Step 2 since φ is satisfiable and hence **Sat**(φ)=**yes**.

Step 2: We set $\mathcal{X} = \{b \wedge c, e \vee f, a \wedge b, \neg e \wedge \neg f \wedge (\neg a \vee \neg b \vee \neg c))\}$ since **Nnf**(φ)=$a \wedge (\Box(b \wedge c) \vee \Box(e \vee f)) \wedge \Diamond(a \wedge b) \wedge \Diamond(\neg e \wedge \neg f \wedge (\neg a \vee \neg b \vee \neg c))$.

Step 3: We check whether there is some subset of $\mathcal{X}$ satisfying conditions (a) and (b). We claim that there is no such subset. To see why, notice that $a \wedge \Box(b \wedge c) \wedge \Diamond(a \wedge b) \wedge \Diamond(\neg e \wedge \neg f \wedge (\neg a \vee \neg b \vee \neg c))$ is the only term in $\mathbf{Dnf}(\varphi)$. Moreover, there is only one set of conjuncts of this term which implies $\Diamond(a \wedge b \wedge c)$, namely $\{\Diamond(a \wedge b), \Box(b \wedge c)\}$. But that means that S must contain either $a \wedge b$ or $b \wedge c$ in order to satisfy condition (b)(i). As $a \wedge b \wedge c$ implies both $a \wedge b$ and $b \wedge c$, we are guaranteed that $a \wedge b \wedge c$ will imply the disjunction of elements in S, thereby falsifying condition (a). It follows that there is no subset of $\mathcal{X}$ satisfying the necessary conditions, so **Test$\Diamond$PI** returns **yes**, which is the desired result.

4.2.7 The algorithm TestPI

We now present our algorithm **TestPI** for testing whether a clause λ is a prime implicate of a formula φ. The first two steps of the algorithm treat the limit cases where λ is not an implicate or where one or both of φ and λ is a tautology or contradiction. In Step 3, we apply equivalence-preserving transformations to λ to make it satisfy the requirements of Theorem 4.2.3. Then in Steps 4, 5, and 6 we use the procedures from Theorems 4.2.8, 4.2.9, and 4.2.12 to test whether the three conditions in Theorem 4.2.3 are verified. If the three tests succeed, then by Theorem 4.2.3, the clause is a prime implicate, so we return **yes**. If some test fails, we return **no** as the clause has been shown not to be a prime implicate.

Algorithm 4.3 TestPI

Input: a clause λ and a formula φ

Output: **yes** if λ is a prime implicate of φ, and otherwise **no**

(1) If **Entails**(φ, λ)=**no**, return **no**.

(2) If **Sat**(φ)=**no**, then return **yes** if **Sat**(λ)=**no** and return **no** otherwise. If **Entails**($\top, \lambda$)=**yes**, then return **yes** if **Entails**($\top, \varphi$)=**yes** and **no** otherwise.

(3) For each disjunct γ of λ, if **Entails**($\lambda \setminus \{\gamma\}, \lambda$)=**yes**, then remove γ from λ. For each $1 \leq i \leq n$, if $Diam_i(\lambda)$ is non-empty, replace each disjunct $\Box_i\chi$ of λ by $\Box_i(\chi \vee \bigvee_{\psi \in Diam_i(\lambda)} \psi)$. Call the resulting clause λ'.

(4) For each $\rho \in Prop(\lambda')$: return **no** if **Entails**($\varphi, \lambda' \setminus \{\rho\}$)=**yes**.

(5) For each disjunct $\Box_i\beta$ of λ': check whether there is $T \in \mathbf{Dnf}(\varphi \wedge \neg(\lambda' \setminus \{\Box_i\beta\}))$ such that **Sat**(T)=**yes** and **Entails**($\beta \wedge \bigwedge_{\psi \in Diam_i(\lambda')} \neg\psi$, $\bigwedge_{\zeta \in Box_i(T)} \zeta$)=**yes**, and return **no** if not.

(6) Return **yes** if **Test$\Diamond$PI**($\Diamond_i(\bigvee_{\psi \in Diam_i(\lambda')} \psi), \varphi \wedge \neg(\lambda' \setminus \{\Diamond_i\psi \mid \psi \in Diam_i(\lambda')\})$) returns **yes** for every i such that $Diam_i(\lambda') \neq \emptyset$, and return **no** otherwise.

We will use the next two lemmas to prove the correctness of our recognition algorithm.

Lemma 4.2.16.
If λ is a clause that is not a prime implicate of φ, then **TestPI** *outputs* ***no*** *on this input.*

Proof. Let us begin by considering a formula λ which is a clause but that is not a prime implicate of φ. There are two possible reasons for this: either λ is not an implicate of φ, or it is an implicate but there exists some stronger implicate. In the first case, we have **Entails**(φ, λ)=**no**, so **TestPI** returns **no** in Step 1, as desired. We will now focus on the case where λ is an implicate but not a prime implicate. We begin by treating the limit cases where one or both of φ and λ is a tautology or contradiction. Given that we know λ to be a non-prime implicate of φ, there are only two possible scenarios: either $\varphi \models \bot$ and $\lambda \not\models \bot$, or $\not\models \varphi$ and $\models \lambda$. In the first case, we have **Sat**(φ)=**no** and **Sat**(λ)=**yes**, and in the second, we have **Entails**$(\top, \varphi)$=**no** and **Entails**$(\top, \lambda)$=**yes**. In both cases, the algorithm returns **no** in Step 2.

If λ is an implicate of φ, and neither φ nor λ is a tautology or contradiction, then the algorithm will continue on to Step 3. In this step, any redundant literals will be deleted from λ, and if λ contains $\Diamond_i$-literals, we add an extra disjunct to the $\Box_i$-literals so that λ satisfies the syntactic requirements of Theorem 4.2.3. Let $\lambda' = \gamma_1 \vee ... \vee \gamma_k \vee \bigvee_{i=1}^{n} (\Diamond_i \psi_{i,1} \vee ... \vee \Diamond_i \psi_{i,l_i} \vee \Box_i \chi_{i,1} \vee ... \vee \Box_i \chi_{i,m_i})$ be the clause at the end of Step 3 once all modifications have been made to λ. As the transformations in Step 3 are equivalence-preserving (Theorem 2.3.1), the clause λ' is equivalent to the original clause λ, so λ' is a non-tautologous non-prime implicate of φ. This means φ and λ' now satisfy all of the conditions of Theorem 4.2.3. It follows then that one of the following holds:

(a) $\gamma_1 \vee ... \vee \gamma_k \notin \Pi(\varphi \wedge \neg(\lambda' \setminus \{\gamma_1, ..., \gamma_k\}))$

(b) $\Box_i (\chi_{i,j} \wedge \neg\psi_{i,1} \wedge ... \wedge \neg\psi_{i,l_i}) \notin \Pi(\varphi \wedge \neg(\lambda' \setminus \{\Box\chi_{i,j}\}))$
for some $1 \leq i \leq n$ and $1 \leq j \leq m_i$

(c) $\Diamond_i (\psi_{i,1} \vee ... \vee \psi_{i,l_i}) \notin \Pi(\varphi \wedge \neg(\lambda' \setminus \{\Diamond_i \psi_{i,1}, ..., \Diamond_i \psi_{i,l_i}\}))$
for some $1 \leq i \leq n$

Suppose that (a) holds. Now $\gamma_1 \vee ... \vee \gamma_k$ is a non-tautologous propositional clause implied by $\varphi \wedge \neg(\lambda' \setminus \{\gamma_1, ..., \gamma_k\})$ which contains no redundant literals. This means that $\varphi \wedge \neg(\lambda' \setminus \{\gamma_1, ..., \gamma_k\})$ and $\gamma_1 \vee ... \vee \gamma_k$ satisfy the conditions of Theorem 4.2.8. According to this theorem, as $\gamma_1 \vee ... \vee \gamma_k \notin \Pi(\varphi \wedge \neg(\lambda' \setminus \{\gamma_1, ..., \gamma_k\})$, then there must be some γ_j such that $\varphi \wedge \neg(\lambda' \setminus \{\gamma_1, ..., \gamma_k\} \models \gamma_1 \vee ... \vee \gamma_{j-1} \vee \gamma_{j+1} \vee \gamma_k$, hence $\varphi \models \lambda' \setminus \{\gamma_j\}$. It follows that for some γ_j we have **Entails**$(\varphi, \lambda' \setminus \{\gamma_j\})$=**yes**, so

the algorithm returns **no** in Step 4.

Suppose next that (b) holds, and let i and j be such that $\Box_i(\chi_{i,j} \wedge \neg\psi_{i,1} \wedge ... \wedge \neg\psi_{i,l_i}) \notin \Pi(\varphi \wedge \neg(\lambda' \setminus \{\Box\chi_{i,j}\}))$. By Theorem 4.2.9, this means that there is no satisfiable $T \in$**Dnf**(φ) such that $\chi_{i,j} \wedge \neg\psi_{i,1} \wedge ... \wedge \neg\psi_{i,l_i} \models \bigwedge_{\zeta \in Box_i(T)} \zeta$, so for every T either **Sat**(T)=**no** or **Entails**$(\chi_{i,j} \wedge \neg\psi_{i,1} \wedge ... \wedge \neg\psi_{i,l_i}, \bigwedge_{\zeta \in Box_i(T)} \zeta)$=**no**. It follows that the algorithm returns **no** in Step 5 while examining the disjunct $\Box_i \chi_{i,j}$.

Finally consider the case where neither (a) nor (b) holds but (c) does, and let i be such that $\Diamond_i(\psi_{i,1} \vee ... \vee \psi_{i,l_i}) \notin \Pi(\varphi \wedge \neg(\lambda' \setminus \{\Diamond_i \psi_{i,1}, ..., \Diamond_i \psi_{i,l_i}\}))$. Then in Step 6, we will call **Test**$\Diamond$**PI**$(\Diamond_i(\psi_{i,1} \vee ... \vee \psi_{i,l_i}), \varphi \wedge \neg(\lambda' \setminus \{\Diamond_i\psi_{i,1}, ..., \Diamond_i\psi_{i,l_i}\}))$. As $\Diamond_i(\psi_{i,1} \vee ... \vee \psi_{i,l_i})$ is not a prime implicate of $\varphi \wedge \neg(\lambda' \setminus \{\Diamond_i \psi_{i,1}, ..., \Diamond_i \psi_{i,l_i}\})$ and we have shown **Test**$\Diamond$**PI** to be correct (Theorem 4.2.12), **Test**$\Diamond$**PI** will return **no**, which means **TestPI** will return **no** as well. As we have covered each of the possible cases, we can conclude that if λ is a clause that is not a prime implicate of φ, then **TestPI** outputs **no**. □

Lemma 4.2.17.
If **TestPI** *outputs* ***no*** *with input (λ, φ) and λ is a clause, then λ is not a prime implicate of φ.*

Proof. There are 5 different ways to return **no** (these occur in Steps 1, 2, 4, 5, and 6). Let us consider each of these in turn. The first way that the algorithm can return **no** is in Step 1 if we find that **Entails**(φ, λ)=**no**, and hence that $\varphi \not\models \lambda$. This is correct since λ cannot be a prime implicate if it is not a consequence of φ. In Step 2, we return **no** if **Sat**(φ)=**no** but **Sat**(λ)=**yes**, or if **Entails**$(\top, \lambda)$=**yes** but **Entails**$(\top, \varphi)$=**no**. In the first case, we have $\varphi \models \bot$ and $\lambda \not\models \bot$, and in the second, $\models \lambda$ and $\not\models \varphi$. In both cases, λ cannot be a prime implicate since there exist stronger implicates (any contradictory clause if $\varphi \models \bot$, and any non-tautologous implicate of φ if $\models \lambda$). In Step 3, we may modify λ, but the resulting clause $\lambda' = \gamma_1 \vee ... \vee \gamma_k \vee \bigvee_{i=1}^{n}(\Diamond_i \psi_{i,1} \vee ... \vee \Diamond_i \psi_{i,l_i} \vee \Box_i \chi_{i,1} \vee ... \vee \Box_i \chi_{i,m_i})$ is equivalent to the original (by Theorem 2.3.1), and so λ' is a prime implicate just in the case that λ was. Now in Step 4, we return **no** if we find some propositional disjunct γ_j in λ' for which **Entails**$(\varphi, \lambda' \setminus \{\gamma_j\})$=**yes**, and hence $\varphi \models \lambda' \setminus \{\gamma_j\}$. Now since in Step 3, we have removed all redundant disjuncts from λ, we can be sure that $\lambda' \setminus \{\gamma_j\}$ is strictly stronger than λ'. So we have $\varphi \models \lambda' \setminus \{\gamma_j\} \models \lambda'$ and $\lambda' \not\models \lambda' \setminus \{\gamma_j\}$, which means that λ', hence λ, is not a prime implicate of φ. We now consider Step 5 of **TestPI**. In this step, we return **no** if for some disjunct $\Box_i\chi_{i,j}$ of λ' there is no term T in **Dnf**$(\varphi \wedge \neg(\lambda' \setminus \{\Box_i\chi_{i,j}\}))$ for which both **Sat**(T)=**yes** and **Entails**$(\chi_{i,j} \wedge \neg\psi_{i,1} \wedge ... \wedge \neg\psi_{i,l_i}, \bigwedge_{\zeta \in Box_i(T)} \zeta)$=**yes**. According to

Theorem 4.2.9, this means that $\Box_i(\chi_{i,j} \wedge \neg\psi_{i,1} \wedge ... \wedge \neg\psi_{i,l_i})$ is not a prime implicate of $\varphi \wedge \neg(\lambda \setminus \{\Box_i \chi_{i,j}\})$, which means that λ', and hence λ, is not a prime implicate of φ by Theorem 4.2.3.

Finally let us consider Step 6. In this step, we return **no** if **Test$\Diamond$PI** returns **no** on input $(\Diamond_i(\psi_{i,1} \wedge ... \wedge \psi_{i,l_i}), \varphi \wedge \neg(\lambda' \setminus \{\Diamond_i\psi_{i,1}, ..., \Diamond_i\psi_{i,l_i}\}))$ for some i. By Theorem 4.2.12, we know that this happens just in the case that $\Diamond_i(\psi_{i,1} \wedge ... \wedge \psi_{i,l_i})$ is not a prime implicate of $\varphi \wedge \neg(\lambda' \setminus \{\Diamond_i\psi_{i,1}, ..., \Diamond_i\psi_{i,l_i}\})$. It follows from Theorem 4.2.3 that λ', and hence λ, is not a prime implicate of φ. □

Theorem 4.2.18.
The algorithm **TestPI** *always terminates, and it returns* ***yes*** *on input (λ, φ) if and only if λ is a prime implicate of φ.*

Proof. The algorithm **TestPI** clearly terminates because Steps 1 to 5 involve a finite number of syntactic operations on λ and a finite number of entailment checks. Moreover, the call to **Test$\Diamond$PI** in Step 6 is known to terminate (Theorem 4.2.12). Correctness and completeness have already been shown in Lemmas 4.2.16 and 4.2.17.□

We demonstrate the functioning of **TestPI** on an example.

Example 4.2.19.
We use **TestPI** to test if the clauses $\lambda_1 = b$, $\lambda_2 = \Box b \vee \Box(e \vee f)$, $\lambda_3 = a \vee \Diamond a$, $\lambda_4 = \Diamond(a \wedge b)$, and $\lambda_5 = \Diamond(a \wedge b \wedge c) \vee \Diamond(a \wedge b \wedge c \wedge f) \vee \Box(e \vee f)$ are prime implicates of $\varphi = a \wedge (\Box(b \wedge c) \vee \Box(e \vee f)) \wedge \Diamond(a \wedge b)$.

λ_1: We output **no** in Step 1 since $\varphi \not\models \lambda_1$, so **Entails**$(\varphi, \lambda_1)$=**no**.

λ_2: We skip Steps 1 and 2 since $\lambda \models \lambda_2$ and neither $\varphi \models \bot$ nor $\models \lambda_2$. In Step 3, we make no changes to λ_2 since it contains no redundant literals nor any $\Diamond$-literals. We skip Step 4 since λ_2 has no propositional disjuncts. In Step 5, we return **no** since the only satisfiable term output by **Dnf** on input $(\varphi \wedge \neg(\lambda_2 \setminus \{\Box b\})$ is $a \wedge \Box(b \wedge c) \wedge \Diamond(a \wedge b) \wedge \Diamond(\neg e \wedge \neg f)$, and we have $b \not\models b \wedge c$.

λ_3: We proceed directly to Step 3 since $\lambda \models \lambda_3$, $\varphi \not\models \bot$, and $\not\models \lambda_3$. No modifications are made to λ_3 in Step 3 as it does not contain any redundant literals or $\Box$-literals. In Step 4, we use **Entails** to test whether or not $\varphi \models \lambda_3 \setminus \{a\}$. As **Entails**$(\varphi, \lambda_3 \setminus \{a\})$=**yes**, we output **no**.

λ_4: Steps 1-5 are all inapplicable, so we skip directly to Step 6. In this step, we call **Test$\Diamond$PI** with as input the clause $\Diamond(a \wedge b)$ and the formula $\varphi \wedge \neg(\lambda_4 \setminus \{\Diamond(a \wedge b)\}) = \varphi$. We have already seen in Example 4.2.14 above that **Test$\Diamond$PI** returns **no** on this input, which means that **TestPI** also returns **no**.

λ_5: We proceed directly to Step 3, where we delete the redundant literal $\Diamond(a \wedge b \wedge c \wedge f)$ and then modify the literal $\Box(e \vee f)$. At the end of this step, we have $\lambda_5 = \Diamond(a \wedge b \wedge c) \vee \Box((e \vee f) \vee (a \wedge b \wedge c))$. Step 4 is not applicable since there are no propositional disjuncts in λ_5. In Step 5, we continue since the only satisfiable term output by **Dnf** on input $\varphi \wedge \neg(\lambda_5 \setminus \{\Box((e \vee f \vee (a \wedge b \wedge c))\})$ is $a \wedge \Box(e \vee f) \wedge \Diamond(a \wedge b) \wedge \Box(\neg a \vee \neg b \vee \neg c)$, and $(e \vee f \vee (a \wedge b \wedge c)) \wedge (\neg a \vee \neg b \vee \neg c) \equiv (e \vee f) \wedge (\neg a \vee \neg b \vee \neg c)$. In Step 6, we return **yes** since we call **Test$\Diamond$PI** on input $(\Diamond(a \wedge b \wedge c), \varphi \wedge \neg(\lambda_5 \setminus \{\Diamond(a \wedge b \wedge c)\}))$, and we have previously shown in Example 4.2.15 that **Test$\Diamond$PI** returns **yes** on this input.

We now show that **TestPI** runs in polynomial space.

Lemma 4.2.20.
The algorithm **TestPI** *provided in Figure 4.3 runs in polynomial space in the length of the input.*

Proof. It is clear that Steps 1 through 5 can be carried out in polynomial space in the length of the input, since they simply involve testing the satisfiability of formulae whose lengths are polynomial in $|\lambda| + |\varphi|$. Step 6 can also be carried out in polynomial space since by Theorem 4.2.13 deciding whether the formula $\Diamond_i(\bigvee_{\psi \in Diam_i(\lambda')} \psi)$ is a prime implicate of $\varphi \wedge \neg(\lambda' \setminus \{\Diamond_i \psi \mid \psi \in Diam_i(\lambda')\})$ takes only polynomial space in $|\Diamond_i(\bigvee_{\psi \in Diam_i(\lambda')} \psi)| + |\varphi \wedge \neg(\lambda' \setminus \{\Diamond_i \psi \mid \psi \in Diam_i(\lambda')\})|$, and hence in $|\lambda| + |\varphi|$. We can thus conclude that the algorithm **TestPI** runs in polynomial space in the length of the input.

As we have already shown that **TestPI** decides prime implicate recognition, it follows that this problem is in PSPACE:

Theorem 4.2.21.
Prime implicate recognition is in PSPACE.

Proof. We have shown in Theorem 4.2.18 that **TestPI** always terminates and returns **yes** whenever the clause is a prime implicate and **no** otherwise. This means that **TestPI** is a decision procedure for prime implicate recognition. Since the algorithm has been shown to run in polynomial space (Lemma 4.2.20), we can conclude that prime implicate recognition is in PSPACE.

By putting together Theorems 4.2.1 and 4.2.21, we obtain a tight complexity bound for the prime implicate recognition task.

Corollary 4.2.22.
Prime implicate recognition is PSPACE*-complete.*

We thus have the positive and somewhat surprising result that the worst-case complexity of prime implicate recognition is the same as that of entailment in $\mathcal{K}_n$.

5

Restricted Consequence Finding

In this chapter, we consider two more nuanced notions of prime implicates: new prime implicates, which allow us to isolate the novel facts which can be derived upon arrival of new information, and signature-bounded prime implicates, which allow us to characterize the consequences of a formula over a given signature. We investigate the properties of both notions and their associated reasoning tasks, leveraging results from previous chapters.

5.1 New prime implicates

When information is added incrementally to a knowledge base, it is natural to want to know what new facts can be derived following the addition of a piece of information. In propositional logic, this motivated the introduction of the notion of new prime implicates (cf. discussion in [Mar00]), which are intended to capture the new consequences of φ upon the addition of some piece of information. We can easily extend this notion to $\mathcal{K}_n$:

Definition 5.1.1 (New prime implicate).
Let φ and ψ be $\mathcal{K}_n$ formulae. A clause λ is said to be a *new prime implicate* of a formula ψ given φ, or simply a *φ-prime implicate* of ψ, if and only if:

1. λ is a prime implicate of $\varphi \wedge \psi$
2. λ is not an implicate of φ

Example 5.1.2.
Consider the formula $\varphi =$

$$(a \vee b) \wedge \Box_1(\neg b \vee c) \wedge (b \vee \Diamond_1 b) \wedge \Diamond_2 a \wedge \Diamond_2 e \\ \wedge \Box_2(b \wedge (a \vee c)) \wedge \Box_2 d$$

from Example 3.2.11. The formulae b and $\neg a$ are the two φ-prime implicates of $\neg a$. There are three φ-prime implicates of $\neg b$: $\neg b$, a, and $\Diamond_1(b \wedge c)$. There is only one φ-prime implicate of $\Diamond_1 \top$, which is $\Diamond_1(\neg b \vee c)$. The unique φ-prime implicate of $\Diamond_2(\neg d \wedge e)$ is $\bot$.

It is also possible to define in an analogous way a notion of new prime implicant. We saw in Chapter 1 that such a notion proves useful in abductive reasoning by allowing us to eliminate explanations which conflict with the background knowledge.

Definition 5.1.3 (New prime implicant).
Let φ and ψ be $\mathcal{K}_n$ formulae. A term κ is a *new prime implicant* of a formula ψ given φ, or simply a *φ-prime implicant* of ψ, if and only if:

1. κ is a prime implicant of $\varphi \vee \psi$
2. κ is not an implicant of φ

In what follows, however, we will restrict our attention to new prime implicates, since the corresponding results for new prime implicants follow by duality.

Theorem 5.1.4.
Every φ-prime implicant of a formula ψ is equivalent to the negation of a $\neg\varphi$-prime implicate of $\neg\psi$, and vice-versa.

Proof. The proof is entirely similar to the proof of **Duality** for standard prime implicates/implicants (Theorem 3.2.5). □

5.1.1 Properties of new prime implicates

The following two theorems clarify the relationship between standard and new prime implicates:

Theorem 5.1.5.
Every φ-prime implicate of ψ is a standard prime implicate of $\varphi \wedge \psi$.

Proof. Follows directly from Definition 5.1.1. □

Theorem 5.1.6.
A clause λ is a standard prime implicate of a non-tautologous formula φ if and only if λ is a $\top$-prime implicate of φ.

Proof. Follows directly from Definition 5.1.1. □

Theorem 5.1.5 allows us to transfer our upper bounds on the size and number of standard prime implicates to new prime implicates.

Theorem 5.1.7.
For any pair of formulae φ and ψ, the length of the smallest clausal representation of a φ-prime implicate of ψ is no more than single exponential in $|\varphi| + |\psi|$.

Proof. Direct consequence of Theorems 4.1.6 and 5.1.5. □

Theorem 5.1.8.
The number of non-equivalent new prime implicates for a given pair of formulae is no more than double exponential in the sum of the lengths of the two formulae.

Proof. Direct consequence of Theorems 4.1.12 and 5.1.5. □

In particular, the latter theorem shows that the number of new prime implicates is always finite modulo equivalence.

Our lower bounds on the size and number of standard prime implicates can also be transferred to new prime implicates via Theorem 5.1.6.

Theorem 5.1.9.
The length of the smallest clausal representation of a new prime implicate of a pair of formulae can be exponential in the sum of the lengths of the formulae.

Proof. Follows directly from Theorems 4.1.7 and 5.1.6. □

Theorem 5.1.10.
The number of non-equivalent new prime implicates of a pair of formulae may be double exponential in the sum of the lengths of the formulae.

Proof. Follows directly from Theorems 4.1.13 and 5.1.6. □

Unsurprisingly, the property **Covering** does not hold for new prime implicates since they only capture part of a formula's implicates. We can, however, show them to satisfy a weaker version of the property:

Theorem 5.1.11.
Every implicate of $\varphi \wedge \psi$ which is not an implicate of φ is entailed by some new prime implicate of ψ given φ.

Proof. Consider formulae φ and ψ, and some clause λ such that $\varphi \wedge \psi \models \lambda$ but $\varphi \not\models \varphi$. By the **Covering** property (Theorem 3.2.8), there is some prime implicate π of $\varphi \wedge \psi$ which implies λ. As $\varphi \not\models \lambda$ and $\pi \models \lambda$, it must also be the case that $\varphi \not\models \pi$. It follows that π is a φ-prime implicate of ψ. □

We also have the following relativized version of **Equivalence**:

Theorem 5.1.12.
The new prime implicates of ψ given φ are equivalent to ψ modulo φ.

Proof. For the first direction, let λ be some prime implicate of $\varphi \wedge \psi$. Now if $\varphi \models \lambda$, we are done. Otherwise, if $\varphi \not\models \lambda$, then λ is a φ-prime implicate of ψ, and hence must be implied by the new prime implicates of ψ given φ. It follows then that $\varphi \wedge \psi$ is implied by the new prime implicates of ψ given φ when taken together with φ. The other direction is immediate since the φ-prime implicates of ψ are all implied by $\varphi \wedge \psi$. □

New prime implicates also satisfy a version of the **Distribution** property:

Theorem 5.1.13.
If λ is a φ-prime implicate of $\psi_1 \vee ... \vee \psi_n$, then there exist φ-prime implicates λ_1, ..., λ_n of ψ_1, ..., ψ_n such that $\lambda \equiv \lambda_1 \vee ... \vee \lambda_n$.

Proof. The proof is entirely similar to the proof of **Distribution** for standard prime implicates (Theorem 3.2.10). □

5.1.2 Generating and recognizing new prime implicates

As the φ-prime implicates of ψ are just the standard prime implicates of $\varphi \wedge \psi$ which are not implied by φ, it follows that one can generate the new prime implicates of ψ given φ by first generating the standard prime implicates of $\varphi \wedge \psi$ then filtering out those which are entailed by φ. The disadvantage of this method is that we generate all of the standard prime implicates of $\varphi \wedge \psi$ even when very few of them are new prime implicates. One way to decrease the number of candidates generated is to use the technique mentioned in Chapter 4 of generating the prime implicates of a disjunction in an iterative manner. Thus, if we were generating the new prime implicates of a formula ψ given the formula φ, we would first rewrite $\varphi \wedge \psi$ as a disjunction of terms $T_1 \vee ... \vee T_n$, and then we would generate the new prime implicates of T_1 given φ, then use them to calculate the new prime implicates of $T_1 \vee T_2$ given φ, and so on until we have the new prime implicates of the entire disjunction. The correctness of this approach follows from the **Distribution** property for new prime implicates (Theorem 5.1.13) together with the fact that the new prime implicates of ψ given φ are the same as the new prime implicates of $\varphi \wedge \psi$ given φ.

As for the complexity of recognizing new prime implicates, it is easy to see that recognizing new prime implicates is no harder than recognizing standard prime implicates.

Theorem 5.1.14.
The problem of recognizing new prime implicates is PSPACE*-complete.*

Proof. To decide whether a clause λ is a new prime implicate of ψ given φ, we simply check whether $\varphi \models \lambda$ and then check whether λ is a prime implicate of $\varphi \wedge \psi$. As both standard prime implicate recognition and entailment can be accomplished in polynomial space (Theorems 2.5.7 and 4.2.22), we have membership in PSPACE. For hardness, we use Theorem 5.1.6 which shows how standard prime implicate recognition can be reduced to new prime implicate recognition. □

5.2 Signature-bounded prime implicates

Another natural restriction is to consider only those consequences which belong to a given signature. This notion has been extensively studied in propositional and first-order logic (cf. [Ino92], [del99], [Mar00]).

Definition 5.2.1 (Signature-bounded prime implicate).
Let $\mathcal{L}$ be a signature. A clause λ is a *signature-bounded prime implicate* of φ with respect to $\mathcal{L}$, or simply an *$\mathcal{L}$-prime implicate* of φ, if and only if:

1. λ is an implicate of φ
2. $sig(\lambda) \subseteq \mathcal{L}$
3. If λ' is an implicate of φ such that $sig(\lambda') \subseteq \mathcal{L}$ and $\lambda' \models \lambda$, then $\lambda \models \lambda'$

Let us illustrate the notion of signature-bounded prime implicates with a quick example:

Example 5.2.2.
Consider the formula $\varphi =$

$$(a \vee b) \wedge \Box_1(\neg b \vee c) \wedge (b \vee \Diamond_1 b) \wedge \Diamond_2 a \wedge \Diamond_2 e \\ \wedge \Box_2(b \wedge (a \vee c)) \wedge \Box_2 d$$

that was introduced in Example 3.2.11. There are no $\{a\}$-prime implicates or $\{1\}$-prime implicates of φ. There is a single $\{2\}$-prime implicate of φ, namely $\Diamond_2 \top$. There is a single $\{1, b\}$-prime implicate: $b \vee \Diamond_1 b$. The three $\{2, a, b\}$-prime implicates of φ are $a \vee b$, $\Diamond_2(a \wedge b)$, and $\Box_2 b$. The three $\{2, a, b, d\}$-prime implicates of φ are $a \vee b$, $\Diamond_2(a \wedge b \wedge d)$, and $\Box_2(b \wedge d)$.

Signature-bounded prime implicants can be defined in the same manner. This notion is useful in abductive reasoning when we want to restrict our attention to only those explanations built from a given set of symbols.

Definition 5.2.3 (Signature-bounded prime implicant).
Let $\mathcal{L}$ be a signature. A term κ is a *signature-bounded prime implicant* of φ with respect to $\mathcal{L}$, or simply an *$\mathcal{L}$-prime implicant* of φ, if and only if:

1. κ is an implicant of φ
2. $sig(\kappa) \subseteq \mathcal{L}$
3. If κ' is an implicant of φ such that $sig(\kappa') \subseteq \mathcal{L}$ and $\kappa \models \kappa'$, then $\kappa' \models \kappa$

However, as the following result demonstrates, it is sufficient to restrict our attention to signature-bounded prime implicates.

Theorem 5.2.4.
Every $\mathcal{L}$-prime implicant of a formula φ is equivalent to the negation of an $\mathcal{L}$-prime implicate of $\neg\varphi$, and vice-versa.

Proof. The proof proceeds analogously to the proof of **Duality** for standard prime implicates/implicants (Theorem 3.2.5). □

5.2.1 Properties of signature-bounded prime implicates

We remark that we can recover the standard notion of prime implicates by setting $\mathcal{L}$ equal to the signature of the formula in question.

Theorem 5.2.5.
Every standard prime implicate of φ is a $sig(\varphi)$-prime implicate of φ.

In propositional logic, every $\mathcal{L}$-prime implicate is also a standard prime implicate. The same is not true for $\mathcal{K}_n$ formulae:

Example 5.2.6.
Consider the formula $\varphi =$

$$(a \vee b) \wedge \Box_1(\neg b \vee c) \wedge (b \vee \Diamond_1 b) \wedge \Diamond_2 a \wedge \Diamond_2 e \\ \wedge \Box_2(b \wedge (a \vee c)) \wedge \Box_2 d$$

from Examples 3.2.11 and 5.2.2. Then $\Diamond_2\top$ is a $\{2\}$-prime implicate of φ, and $b \vee \Diamond_1 b$ is a $\{1, b\}$-prime implicate of φ, but neither clause is a standard prime implicate of φ.

There are however some weaker relationships holding between standard and $\mathcal{L}$-prime implicates:

Theorem 5.2.7.
Every $\mathcal{L}$-prime implicate of a formula φ is equivalent to some prime implicate of an $\mathcal{L}$-interpolant of φ, and vice-versa.

Proof. For the first direction, consider some $\mathcal{L}$-prime implicate λ of φ and some $\mathcal{L}$-interpolant ψ of φ. As $sig(\lambda) \subseteq \mathcal{L}$, it must be the case that ψ implies λ. But then by **Covering** (Theorem 3.2.8), there must be some standard prime implicate π of ψ such that $\pi \models \lambda$. Because of Theorem 4.1.5, we can assume that π has signature in $\mathcal{L}$. It follows then that $\lambda \models \pi$, hence $\lambda \equiv \pi$.

For the second direction, let π be a prime implicate of an $\mathcal{L}$-interpolant ψ of φ. Because of Theorem 4.1.5, we know that π is equivalent to some clause π' with signature in $\mathcal{L}$. Now let λ be a clause such that $\varphi \models \lambda \models \pi'$ and $sig(\lambda) \subseteq \mathcal{L}$. As $sig(\lambda) \subseteq \mathcal{L}$, it must be the case that $\psi \models \lambda$. Moreover, we know that π (and hence π') is a prime implicate of ψ, so $\pi' \models \lambda$. It follows that π' is an $\mathcal{L}$-prime implicate of φ, which means π is equivalent to some $\mathcal{L}$-prime implicate of φ. □

Theorem 5.2.8.
Every $\mathcal{L}$-prime implicate of a formula φ is equivalent to the $\mathcal{L}$-interpolant of some prime implicate of φ.

Proof. Let λ be an $\mathcal{L}$-prime implicate of φ. Because of the **Covering** property (Theorem 3.2.8), we can find some prime implicate π of φ such that $\pi \models \lambda$. Now let π' be the $\mathcal{L}$-interpolant of π which is computed by **LangInt**. We know from Theorem 2.6.9 that π' is a clause. As $\pi \models \lambda$ and $sig(\lambda) \subseteq \mathcal{L}$, it follows that $\pi' \models \lambda$, and hence $\lambda \models \pi'$, so $\lambda \equiv \pi'$. □

The converse of Theorem 5.2.8 does not hold as there are prime implicates whose $\mathcal{L}$-interpolants are not $\mathcal{L}$-prime implicates, as the following example demonstrates.

Example 5.2.9.
The clause $\Diamond_1 a$ is the $\{a, c, 1\}$-interpolant of the prime implicate $\Diamond_1(a \wedge b)$ of $\varphi = \Diamond_1(a \wedge b) \wedge \Diamond_1(a \wedge c)$, but $\Diamond_1 a$ is not an $\{a, c, 1\}$-prime implicate of φ.

One consequence of Theorem 5.2.8 is that the number of $\mathcal{L}$-prime implicates is bounded above by the number of prime implicates.

Theorem 5.2.10.
The number of non-equivalent $\mathcal{L}$-prime implicates of a formula is no more than double exponential in the length of the formula.

Proof. Direct consequence of Theorems 4.1.12 and 5.2.8. □

We can also transfer our upper bound on the size of standard prime implicates to signature-bounded prime implicates.

Theorem 5.2.11.
The length of the smallest clausal representation of an $\mathcal{L}$-prime implicate of a formula is no more than single exponential in the length of the formula.

Proof. Let λ be an $\mathcal{L}$-prime implicate of a formula φ. By **Covering** (Theorem 3.2.8), we know that there must be some standard prime implicate π of φ such that $\pi \models \lambda$. From the proof of Theorem 4.1.6, we can assume without loss of generality that π is a clause with at most $2^{|\varphi|}$ disjuncts, each having length at most $2|\varphi|$. Now we can apply the function **LangInt** to π to obtain a formula π'. By Theorem 2.6.6 and Lemma 2.6.9, we know that π' is a clause and the $\mathcal{L}$-interpolant of π. Since $\pi \models \lambda$ and $sig(\lambda) \subseteq \mathcal{L}$ and π' is the $\mathcal{L}$-interpolant of π, it follows that $\pi' \models \lambda$. But λ is assumed to be an $\mathcal{L}$-prime implicate of φ, so we must also have $\lambda \equiv \pi'$.

It remains to be shown that the length of π' is only single exponential in $|\varphi|$. We remark that the function **LangInt** treats each of the disjuncts of π separately. Moreover, by Theorem 2.6.7, the output of **LangInt** is single exponential in the size of the input formula. In other words, we can find some polynomial function f such that the output of **LangInt** on input ψ has length at most $2^{f(|\psi|)}$. But that means that the formula π' has length at most $2^{|\varphi|} * (2^{f(2|\varphi|)} + 1)$, since there are no more than $2^{|\varphi|}$ disjuncts, each having size at most $2^{f(2|\varphi|)}$ (the extra 1 is for the disjunction symbols connecting the disjuncts). This proves the result since this expression is clearly single exponential in $|\varphi|$. □

Our lower bounds on the number and size of standard prime implicates also carry over to $\mathcal{L}$-prime implicates thanks to Theorem 5.2.5.

Theorem 5.2.12.
The number of non-equivalent $\mathcal{L}$-prime implicates of a formula may be double exponential in the length of the formula.

Proof. Follows directly from Theorems 4.1.13 and 5.2.5. □

Theorem 5.2.13.
The length of the smallest clausal representation of an $\mathcal{L}$-prime implicate of a formula can be exponential in the length of the formula.

Proof. Follows directly from Theorems 4.1.7 and 5.2.5. □

Using Theorem 5.2.7, we can infer the following proposition which can be seen as a weaker version of the **Covering** property.

Theorem 5.2.14.
Every implicate of φ with signature contained in $\mathcal{L}$ is entailed by some $\mathcal{L}$-prime implicate of φ.

Proof. Let φ be a formula, and let λ be some implicate of φ with $sig(\lambda) \subseteq \mathcal{L}$. We know that λ must be implied by every $\mathcal{L}$-interpolant of φ, and therefore by **Covering** (Theorem 3.2.8), we can find some prime implicate π of an $\mathcal{L}$-interpolant of φ such that $\pi \models \lambda$. According to Theorem 5.2.7, there must be some $\mathcal{L}$-prime implicate π' of φ which is equivalent to π, which means we have found a $\mathcal{L}$-prime implicate of φ which entails λ. □

A weaker version of **Equivalence** holds as well:

Theorem 5.2.15.
The set of $\mathcal{L}$-prime implicates of a formula is equivalent to the $\mathcal{L}$-interpolant of the formula.

Proof. For the first direction, let φ be a formula, and let π be some prime implicate of the $\mathcal{L}$-interpolant of φ'. By Theorem 5.2.7, the clause π is equivalent to, and hence implied by, some $\mathcal{L}$-prime implicate of φ. Using the **Equivalence** property for standard prime implicates (Theorem 3.2.9), we find that the set of $\mathcal{L}$-prime implicates of φ implies the $\mathcal{L}$-interpolant of φ. The other direction is immediate since the $\mathcal{L}$-interpolant of φ must by definition imply each of the $\mathcal{L}$-prime implicate of φ. □

The **Distribution** property can also be formulated for $\mathcal{L}$-prime implicates:

Theorem 5.2.16.
If λ is an $\mathcal{L}$-prime implicate of $\varphi_1 \vee ... \vee \varphi_n$, then there exist $\mathcal{L}$-prime implicates $\mu_1, ..., \mu_n$ of $\varphi_1, ..., \varphi_n$ such that $\lambda \equiv \mu_1 \vee ... \vee \mu_n$.

Proof. Very similar to the proof for standard prime implicates (Theorem 3.2.10).□

5.2.2 Generating signature-bounded prime implicates

There are a couple of different ways of exploiting **GenPI** in the computation of signature-bounded prime implicates. A first possibility would be to take advantage of Theorem 5.2.8 which tells us that the $\mathcal{L}$-prime implicates of a formula are the logically strongest clauses among the $\mathcal{L}$-interpolants of the formula's prime implicates.

This means that we can generate the $\mathcal{L}$-prime implicates of a formula by first using **GenPI** to obtain the formula's prime implicates, then taking the $\mathcal{L}$-interpolants of the prime implicates, and finally comparing the resulting clauses to isolate the logically strongest ones. Another possibility would be to replace the input formula by its $\mathcal{L}$-interpolant and then call **GenPI** to generate the prime implicates of the $\mathcal{L}$-interpolant. Because of Theorem 5.2.7, we know that the clauses outputted by **GenPI** will be exactly the $\mathcal{L}$-prime implicates of the original formula.

Both of the above methods yield $\mathcal{L}$-prime implicates which are at most single-exponentially larger than the input formula. This was shown for the first method in the proof of Theorem 5.2.11. For the second method, we use the fact that $\mathbf{LangInt}(\varphi, \mathcal{L})$ is already a disjunction of terms, so applying **Dnf** to $\mathbf{LangInt}(\varphi, \mathcal{L})$ does not increase its length. This means that there will be only single-exponentially many disjuncts in each clauses in CANDIDATES, and that the disjuncts of these clauses will be at most single-exponentially large.

Is there any reason to prefer one method over the other? In fact, there is. We remark that in the second method, we can eliminate weaker elements in CANDIDATES by using standard prime implicate recognition, whereas with the first method, we need to perform $\mathcal{L}$-prime implicate recognition (or resort to a pairwise comparison of the elements in CANDIDATES). As we shall see in the following section, $\mathcal{L}$-prime implicate recognition is of higher complexity than standard prime implicate recognition, leading us to privilege the second generation strategy.

5.2.3 Recognizing signature-bounded prime implicates

We know from Example 5.2.6 that $\mathcal{L}$-prime implicates may not be standard prime implicates, which means that the PSPACE-completeness result for standard prime implicate recognition is not much help to us. Indeed, it turns out that $\mathcal{L}$-prime implicate recognition is considerably more difficult computationally than standard prime implicate recognition.We can show this task to be CONEXPTIME-hard:

Theorem 5.2.17.
$\mathcal{L}$-prime implicate recognition is CONEXPTIME*-hard.*

Proof. The proof is via a reduction of the conservative extension decision problem for $\mathcal{K} = \mathcal{K}_1$ formulae to the $\mathcal{L}$-prime implicate recognition problem. We recall that a formula $\varphi_1 \wedge \varphi_2$ is a conservative extension of φ_1 if and only if for every formula ψ with $var(\psi) \subseteq var(\varphi_1)$ we have $\varphi_1 \wedge \varphi_2 \models \psi$ only if $\varphi_1 \models \psi$. We will show that $\varphi_1 \wedge \varphi_2$ is a conservative extension of φ_1 if and only if $\Diamond_1\mathbf{Nnf}(\varphi_1)$ is a $var(\varphi_1) \cup \{1\}$-prime implicate of $\Diamond_1(\varphi_1 \wedge \varphi_2)$. As the conservative extension

decision problem for $\mathcal{K}$ formulae was proven CoNExptime-complete in [GLWZ06], it follows that $\mathcal{L}$-prime implicate recognition must be CoNExptime-hard.

For the first direction, let us suppose that $\varphi_1 \wedge \varphi_2$ is a conservative extension of φ_1. It follows that φ_1 is a $var(\varphi_1) \cup \{1\}$-interpolant of $\varphi_1 \wedge \varphi_2$. Using Lemma 2.6.10, we then find that $\Diamond_1\varphi_1$ is a $var(\varphi_1) \cup \{1\}$-interpolant of $\Diamond_1(\varphi_1 \wedge \varphi_2)$. That means that if λ is a clause such that $sig(\lambda) \subseteq var(\varphi_1) \cup \{1\}$ and $\Diamond_1(\varphi_1 \wedge \varphi_2) \models \lambda \models \Diamond_1\varphi_1$, we must also have $\Diamond_1\varphi_1 \models \lambda$. This means that the clause $\Diamond_1\mathbf{Nnf}(\varphi_1) \equiv \Diamond_1\varphi_1$ must be a $var(\varphi_1) \cup \{1\}$-prime implicate of the formula $\Diamond_1(\varphi_1 \wedge \varphi_2)$.

For the other direction, suppose that $\Diamond_1\mathbf{Nnf}(\varphi_1)$ is a $var(\varphi_1) \cup \{1\}$-prime implicate of $\Diamond_1(\varphi_1 \wedge \varphi_2)$. That means that for every clause λ with $sig(\lambda) \subseteq var(\varphi_1) \cup \{1\}$ and $\Diamond_1(\varphi_1 \wedge \varphi_2) \models \lambda \models \Diamond_1\mathbf{Nnf}(\varphi_1)$, we have $\Diamond_1\mathbf{Nnf}(\varphi_1) \models \lambda$. In particular, if $\Diamond_1\psi$ is a clause with signature in $var(\varphi_1) \cup \{1\}$ such that $\Diamond_1(\varphi_1 \wedge \varphi_2) \models \Diamond_1\psi \models \Diamond_1\mathbf{Nnf}(\varphi_1)$, then $\Diamond_1\mathbf{Nnf}(\varphi_1) \models \Diamond_1\psi$. It follows then from Theorem 2.3.1 and the fact that **Nnf** is equivalence- and signature-preserving (Theorem 2.4.2) that for every formula ψ which is implied by $\varphi_1 \wedge \varphi_2$ and with $sig(\psi) \subseteq var(\varphi_1) \cup \{1\}$, we have $\varphi_1 \models \psi$, i.e. φ_1 is a $var(\varphi_1) \cup \{1\}$-interpolant of $\varphi_1 \wedge \varphi_2$. It follows that $\varphi_1 \wedge \varphi_2$ is a conservative extension of φ_1. □

We now provide an Expspace upper bound. Our proof makes reference to the algorithm **TestLangPI** defined below:

Algorithm 5.1 TestLangPI

Input: a formula φ, a clause λ, and a signature $\mathcal{L}$

Output: **yes** is λ is not an $\mathcal{L}$-prime implicate of φ, and **no** otherwise

(1) If **Entails**(φ, λ)=**no** or $sig(\lambda) \not\subseteq \mathcal{L}$, return **yes**.
(2) Guess some clause π of length at most $2^{|\varphi|} * (2^{f(2|\varphi|)} + 1)$ with signature in $\mathcal{L}$.
(3) If **Entails**(φ, π)=**no** or **Entails**(π, λ)=**no**, then return **no**.
(4) If **Entails**(λ, π)=**no**, return **yes**. Otherwise, return **no**.

Note: in Step 2, we let f be some function such that $|\mathbf{LangInt}(\psi, \mathcal{L})| \leq 2^{f(|\psi|)}$ on every input $(\psi, \mathcal{L})$. The existence of such a function is guaranteed by Corollary 2.6.8.

Theorem 5.2.18.
$\mathcal{L}$-prime implicate recognition is in Expspace.

Proof. We will show that the non-deterministic algorithm **TestLangPI** decides the complement of the $\mathcal{L}$-prime implicate recognition problem, and moreover that

it runs using only single exponential space. This is sufficient to prove the result since CoNExpspace=Expspace.

We start by showing the correctness of our procedure. First suppose that λ is not an $\mathcal{L}$-prime implicate of φ. Then either λ is not an implicate of φ, or it does not have signature in $\mathcal{L}$, or there is some $\mathcal{L}$-prime implicate ζ of φ such that $\zeta \models \lambda \not\models \zeta$. In the first two cases, the algorithm will return **yes** in Step 1. In the third case, we proceed to Step 2 where we guess a clause of length at most $2^{|\varphi|} * (2^{f(2|\varphi)|} + 1)$ and with signature in $\mathcal{L}$. We know from the proof of Theorem 5.2.11 that every $\mathcal{L}$-prime implicate of φ must be equivalent to some clause with signature in $\mathcal{L}$ and with length at most $2^{|\varphi|} * (2^{f(2|\varphi)|} + 1)$. It follows then that in Step 2 we can choose the clause π so that $\pi \equiv \zeta$, which means we will satisfy the tests in Step 3 and proceed on to Step 4. In this step, we test whether $\lambda \not\models \pi$. As we know that $\lambda \not\models \zeta$ and $\zeta \equiv \pi$, we must also have $\lambda \not\models \pi$, so the algorithm will return **yes** in Step 5.

Next suppose that λ is an $\mathcal{L}$-prime implicate of φ. Then the tests in Step 1 will not succeed, and we will go directly to Step 2, where we guess some clause π of length at most $2^{|\varphi|} * (2^{f(2|\varphi)|} + 1)$ and with signature in $\mathcal{L}$. If π does not satisfy the required conditions, then we will output **no** in Step 3. Otherwise, in Step 4, we will test whether $\lambda \not\models \pi$. Now since λ is an $\mathcal{L}$-prime implicate of φ and π is a clause with signature in $\mathcal{L}$ such that $\varphi \models \pi \models \lambda$, it follows that $\lambda \models \pi$, so we will return **no**.

Now we consider the spatial complexity of **TestLangPI**. The first step runs in polynomial space in $|\varphi| + |\lambda|$ by Theorem 2.5.7. The second step takes single-exponential space since we guess a clause of length at most $2^{|\varphi|} * (2^{f(2|\varphi)|} + 1)$ (and f is assumed to be a polynomial function). Steps 3 and 4 also require at most single-exponential space since we are performing entailment tests on formulae with length at most single-exponentially larger than $|\varphi| + |\lambda|$. □

The exact complexity $\mathcal{L}$-prime implicate recognition is currently unknown, but we conjecture that the problem is CoNExptime-complete.

6

Prime Implicate Normal Form

In this chapter, we introduce a normal form for $\mathcal{K}_n$ formulae which is based upon the notion of prime implicate studied in the previous chapters. We investigate the properties of our normal form, showing in particular that entailment between formulae in prime implicate normal form can be carried out in quadratic time using a simple structural comparison algorithm. We also show that uniform interpolation is tractable for formulae in our normal form. Afterwards, we propose an algorithm for putting concepts into prime implicate normal form, and we investigate the spatial complexity of this transformation, showing there to be an at most double exponential blowup in formula size. At the end of the chapter, we compare our normal form to other normal forms previously proposed in the literature.

6.1 Motivation

As we mentioned in Chapter 1, knowledge compilation is a technique for dealing with the high complexity of reasoning which consists in a preliminary off-line phase in which a knowledge base is transformed into an equivalent base which allows for tractable reasoning, followed by a second online phase in which reasoning is performed on the compiled knowledge base. One well-known target language for knowledge compilation in propositional logic is prime implicate normal form, in which a formula is represented as the conjunction of its prime implicates. A natural idea would be to use our selected definition of prime implicate to define in an analogous manner a notion of prime implicate normal form for $\mathcal{K}_n$ formulae. Unfortunately, the normal form we obtain satisfies few of the nice properties of the propositional case. For instance, we find that entailment between two $\mathcal{K}_n$ formulae

in prime implicate normal form is no easier than between arbitrary $\mathcal{K}_n$ formulae. To see why, consider any pair of formulae φ_1 and φ_2 in negation normal form. The formulae $\Diamond\varphi_1$ and $\Diamond\varphi_2$ are their own prime implicates and hence would be in prime implicate normal form if we used the naïve definition. As φ_1 entails φ_2 just in the case that $\Diamond\varphi_1$ entails $\Diamond\varphi_2$, we can reduce entailment between arbitrary formulae in NNF to entailment between formulae in prime implicate normal form. As the former problem is known to be PSPACE-complete (by Corollary 2.5.2), it follows that the latter is PSPACE-complete as well.

This appears to be quite a disappointing result as one would hope that the computational difficulty of representing a formula by its prime implicates would be offset by some good computational properties of the resulting formula. As it turns out, however, the problem lies not in our definition of prime implicates but rather in the naïve way of defining prime implicate normal form. Indeed, in this chapter, we propose a more sophisticated definition of prime implicate normal form, which takes as its basis our selected notion of prime implicate but places some additional restrictions on the way the prime implicates are represented.

6.2 Definition of Prime Implicate Normal Form

The disappointing behavior of the naïve definition of prime implicate normal form appears to stem at least partly from the fact that the formulae behind the modalities are left undecomposed. It seems then that we should require not only that the original formula be represented by its prime implicates but also that the sub-formulae appearing in the prime implicates be themselves represented by their prime implicates. This intuition is at the heart of our definition of prime implicate normal form for $\mathcal{K}_n$ formulae:

Definition 6.2.1 (Prime Implicate Normal Form).
A formula φ is in *prime implicate normal form* if and only if it satisfies one of the following conditions:

1. $\varphi = \bot$
2. $\varphi = \top$
3. $\varphi \not\models \bot$ and $\not\models \varphi$ and $\varphi = \lambda_1 \wedge ... \wedge \lambda_p$ where
   (a) $\lambda_i \not\models \lambda_j$ for $i \neq j$
   (b) each prime implicate of φ is equivalent to some conjunct λ_i
   (c) every λ_i is such that
      i. if θ is a disjunct of λ_i, then $\lambda_i \not\equiv \lambda_i \setminus \{\theta\}$

ii. $|Diam_k(\lambda_i)| \leq 1$ for every $1 \leq k \leq n$

iii. if $\psi \in Diam_k(\lambda_i) \cup Box_k(\lambda_i)$ for some $1 \leq k \leq n$, then ψ is in prime implicate normal form

iv. if $\psi \in Diam_k(\lambda_i)$ and $\zeta \in Box_k(\lambda_i)$ for some $1 \leq k \leq n$, then $\psi \models \zeta$

Let us briefly go over the different points of the definition. The first two items state that all unsatisfiable formulae must be represented as $\bot$ and all tautologous formulae must be represented as $\top$. All other formulae are to be represented by a conjunction of their prime implicates, but we place some strong restrictions on how the prime implicates themselves are represented. First, we require that they contain no unnecessary disjuncts (part (i) of 3c). We also stipulate that they contain at most one $\Diamond_k$-disjunct for each k (part (ii)) and that the formulae appearing behind the modalities be themselves in prime implicate normal form (part (iii)). Finally, we demand that if a prime implicate contains disjuncts $\Diamond_k \psi$ and $\Box_k \zeta$ then ψ and ζ are such that $\psi \models \zeta$ (part (iv)). This requirement may seem a little less intuitive than the others, but it ensures that if a $\Box_k$-formula entails a clause, then it entails some $\Box_k$-formula appearing in the clause [1]. This property is crucial since it will allow our algorithm for entailment-testing to treat the universal modalities separately from the existential ones.

We remark that in the case of propositional formulae, our definition of prime implicate normal form coincides with the standard propositional definition.

Example 6.2.2.
Some examples of clauses which are not in prime implicate normal form:

- $\lambda_1 = \Box_1 b \vee \Diamond_1 c$, since $c \not\models b$
- $\lambda_2 = \Diamond_1(b \wedge \Box_2 \bot) \vee \Diamond_1(c \vee d) \vee a$, since $|Diam_1(\lambda_2)| = 2$
- $\lambda_3 = \Diamond(a \wedge \neg a)$ since $\lambda_3 \models \bot$ but $\lambda_3 \neq \bot$
- $\lambda_4 = \Box_1(a \vee \Box_2(b \vee \neg b))$ since $\models \lambda_4$ but $\lambda_4 \neq \top$
- $\lambda_5 = a \vee \Box_1(a \wedge b) \vee \Box_1(a \wedge b \wedge \neg c)$, since $\lambda_5 \equiv \lambda_5 \setminus \{\Box_1(a \wedge b \wedge \neg c)\}$
- $\lambda_6 = \Box_1((a \wedge b) \vee c)$ since $(a \wedge b) \vee c$ is not in prime implicate normal form

Example 6.2.3.
Some examples of general formulae which are not in prime implicate normal form:

- $\varphi_1 = (a \vee \Diamond_2 c) \wedge (\neg a \vee c)$, since the prime implicate $c \vee \Diamond_2 c$ is not equivalent to any conjunct of φ_1
- $\varphi_2 = a \wedge (a \vee \Box_3 b)$, since $a \models (a \vee \Box_3 b)$
- $\varphi_3 = (a \vee \neg d) \wedge \Box_1((a \wedge b) \vee c)$, since the subformula $(a \wedge b) \vee c$ of φ_3 is not in prime implicate normal form

1. This does not hold in general: $\Box a \models \Diamond a \vee \Box b$ but $\Box a \not\models \Box b$.

Example 6.2.4.
Some examples of clauses which are in prime implicate normal form:

- $a \vee \Box_2 \Box_1 d$, since:
  - no unnecessary disjuncts, nor any $\Diamond$-disjuncts
  - d is in prime implicate normal form, and hence so is $\Box_1 d$
- $\neg a \vee \Diamond_2 (a \wedge b)$, since:
  - no unnecessary disjuncts or $\Box$-disjuncts
  - a single $\Diamond_2$-disjunct
  - $a \wedge b$ is in prime implicate normal form
- $\Box_1 ((a \vee \Diamond_2 \top) \wedge (\Box_2 \bot \vee c) \wedge (a \vee c))$, since:
  - consists of a single $\Box$-literal, so satisfies trivially conditions 3(c)(i), 3(c)(ii), and 3(c)(iv)
  - the subformula $a \vee \Diamond_2 \top$ is in prime implicate normal form since it contains no unnecessary disjuncts or $\Box$-disjuncts, a single $\Diamond_2$-disjunct, and $\top$ is in prime implicate normal form
  - the subformula $\Box_2 \bot \vee c$ is in prime implicate normal form since it contains no unnecessary disjuncts nor $\Diamond$-disjuncts, and $\bot$ is in prime implicate normal form
  - $(a \vee \Diamond_2 \top) \wedge (\Box_2 \bot \vee c) \wedge (a \vee c)$ is in prime implicate normal form since it contains no unnecessary conjuncts, and its conjuncts are its only prime implicates and are themselves in prime implicate normal form (see previous two bullets)
- $\Box_2 ((\Box_1 d \vee a) \wedge (\Box_1 d \vee b)) \vee \Diamond_2 (a \wedge b)$, since:
  - no unnecessary disjuncts
  - only a single $\Diamond_2$-disjunct
  - $a \wedge b$ is in prime implicate normal form
  - $(\Box_1 d \vee a) \wedge (\Box_1 d \vee b)$ is in prime implicate normal form, since its conjuncts are its only prime implicates, there are no redundant conjuncts, and both conjuncts are themselves in prime implicate normal form
  - we have $a \wedge b \models (\Box_1 d \vee a) \wedge (\Box_1 d \vee b)$

Example 6.2.5.
Consider the formula φ defined as follows

$$
\begin{aligned}
& (a \vee \Box_2 \Box_1 d) \\
\wedge \quad & (\neg a \vee \Diamond_2 (a \wedge b)) \\
\wedge \quad & (\Box_2 ((\Box_1 d \vee a) \wedge (\Box_1 d \vee b)) \vee \Diamond_2 (a \wedge b)) \\
\wedge \quad & \Box_1 ((a \vee \Diamond_2 \top) \wedge (\Box_2 \bot \vee c) \wedge (a \vee c)) \\
\wedge \quad & \Diamond_1 (a \wedge \neg c \wedge \Box_2 \bot)
\end{aligned}
$$

We would like to show that φ is in prime implicate normal form. We first note that φ is neither a contradiction nor a tautology, so part 3 of Definition 6.2.1 applies. We then note that every prime implicate of φ is equivalent to a conjunct of φ. This is because φ is equivalent to the formula in Example 4.1.16, and each of the prime implicates from Example 4.1.16 is equivalent to one of the conjuncts of φ. Moreover, none of the conjuncts of φ entails one of the other conjuncts, as can easily be verified. Finally, we know from Example 6.2.4 that the first four conjuncts of φ satisfies conditions 3(c)(i)-3(c)(iv), and the same can be shown for its final conjunct $\Diamond_1 (a \wedge \neg c \wedge \Box_2 \bot)$ (since $\Box_2 \bot$, and hence $a \wedge \neg c \wedge \Box_2 \bot$, is in prime implicate normal form).

We will show later in the chapter (Theorem 6.4.3) that Definition 6.2.1 that every formula can be rewritten as an equivalent formula in prime implicate normal form. We first motivate the interest of doing so by exhibiting some of the desirable properties of formulae in prime implicate normal form.

6.3 Properties of Prime Implicate Normal Form

In this section, we show that prime implicate normal form has some nice properties which make it an interesting target language for knowledge compilation.

6.3.1 Tractable entailment

The most important criterion when selecting a normal form for knowledge compilation is the set of polynomial time queries that the normal form supports. In [DM02], the authors enumerate a set of queries which they then use to compare different normal forms for propositional logic. Of the eight queries they consider, only four are well-defined [2] for $\mathcal{K}_n$: satisfiability-testing, tautology-testing, entailment, and equivalence-testing. We show that for formulae in prime implicate normal form, all four queries are computable in polynomial time.

For satisfiability and tautology-testing, there is really nothing to prove since by definition a formula φ in prime implicate normal form is unsatisfiable just in the case that $\varphi = \bot$ and is tautologous just in the case that $\varphi = \top$. It follows that these tasks can be carried out in constant-time.

For entailment and equivalence, we introduce a structural comparison algorithm **Π-Entail** which decides entailment between formulae in prime implicate normal

2. For example, clausal entailment is under-specified since there are many possible definitions of clauses in $\mathcal{K}_n$, and model counting makes little sense since every formula has infinitely many distinct models.

Algorithm 6.1 Π**-Entail**

Input: formulae φ_1 and φ_2, both in prime implicate normal form

Output: **yes** if $\varphi_1 \models \varphi_2$, and **no** otherwise

(1) If $\varphi_1 = \bot$ or $\varphi_2 = \top$, return **yes**.

(2) If $\varphi_1 = \top$ and $\varphi_2 \neq \top$ or $\varphi_2 = \bot$ and $\varphi_1 \neq \bot$, return **no**.

(3) For each conjunct λ of φ_2

Set $MatchFound = no$

For each conjunct θ of φ_1

If $MatchFound = no$, then set $MatchFound = yes$ if the following three conditions are satisfied:

(a) $Prop(\theta) \subseteq Prop(\lambda)$

(b) if $\psi \in Diam_k(\theta)$, then there is $\psi' \in Diam_k(\lambda)$ such that Π-**Entail**(ψ, ψ')=**yes**

(c) if $\psi \in Box_k(\theta)$, then there is some $\psi' \in Box_k(\lambda)$ such that Π-**Entail**(ψ, ψ')=**yes**

If $MatchFound = no$, return **no**.

Return **yes**.

form. Let us explain briefly the functioning of Π-**Entail**. The first two steps treat limit cases where one or both of the formulae is unsatisfiable or tautologous. For all other pairs of formulae, we proceed to Step 3, in which we perform a structural comparison of the two formulae. We know that a formula φ_1 entails a formula φ_2 in prime implicate normal form just in the case that φ_1 entails each of the conjuncts of φ_2. Moreover, it follows from the **Covering** property (Theorem 3.2.8) that φ_1 entails a clausal conjunct λ of φ_2 if and only if some prime implicate of φ_1 entails λ. As formulae in prime implicate normal form are conjunctions of their prime implicates, testing whether φ_1 entails φ_2 comes down to testing whether each conjunct of φ_2 is entailed by some conjunct of φ_1. If we hadn't placed any requirements on the form of the conjuncts, then this problem would be as hard as entailment in general. But since φ_1 and φ_2 are in prime implicate normal form, their conjuncts have a particular structure which makes subsumption easy to test. We first check that the propositional literals in the first conjunct all appear in the second conjunct. We then call Π-**Entail** on sub-formulae appearing in the two conjuncts in order to ensure that each $\Diamond$- or $\Box$-formula appearing in the first conjunct entails some $\Diamond$- or $\Box$-formula in the second. The algorithm performs these checks on each possible pair of conjuncts and returns **no** if it finds some conjunct of φ_2 which does not subsume any conjunct of φ_1. If no such conjunct is found, the algorithm returns **yes** since every conjunct of φ_2 has been shown to be implied

by some conjunct of φ_1, which means that φ_1 entails φ_2.

We illustrate the functioning of the algorithm on an example:

Example 6.3.1.
Let φ be defined as in Example 6.2.5:

$$\begin{aligned} & (a \vee \Box_2 \Box_1 d) \\ \wedge \quad & (\neg a \vee \Diamond_2 (a \wedge b)) \\ \wedge \quad & (\Box_2 ((\Box_1 d \vee a) \wedge (\Box_1 d \vee b)) \vee \Diamond_2 (a \wedge b)) \\ \wedge \quad & \Box_1 ((a \vee \Diamond_2 \top) \wedge (\Box_2 \bot \vee c) \wedge (a \vee c)) \\ \wedge \quad & \Diamond_1 (a \wedge \neg c \wedge \Box_2 \bot) \end{aligned}$$

and let ψ be the formula

$$(a \vee b \vee \Box_2 a \vee \Box_2 \Box_1 (d \vee \Diamond_2 a)) \wedge (\neg c \vee \Diamond_1 (a \wedge b))$$

We showed in Example 6.2.5 that φ is in prime implicate normal form, and it can be verified that this is also the case for ψ. We can thus use **Π-Entail** to decide whether $\varphi \models \psi$. The algorithm will proceed directly to Step 3 as neither φ nor ψ is equal to $\top$ or $\bot$.

In Step 3, we consider each of the conjuncts of ψ in turn. We start with the first conjunct of ψ which is $a \vee b \vee \Box_2 a \vee \Box_2 \Box_1 (d \vee \Diamond_2 a)$. The variable $MatchFound$ is initialized to *no*, and we then select the first conjunct of φ which is $a \vee \Box_2 \Box_1 d$. This pair of conjuncts satisfies condition (a) since $\{a\} \subseteq \{a, b\}$. Condition (b) is trivially satisfied since there are no $\Diamond$-disjuncts in $a \vee \Box_2 \Box_1 d$. To determine whether condition (c) holds for the pair of conjuncts, we need to check whether either **Π-Entail**($\Box_1 d$, a)=**yes** or **Π-Entail**($\Box_1 d$, $\Box_1 (d \vee \Diamond_2 a)$)=**yes**. We have **Π-Entail**(d, a)=**no** since the two input formulae consist of a single clause, and $\{d\} \not\subseteq \{a\}$. To determine the value of **Π-Entail**($\Box_1 d$, $\Box_1 (d \vee \Diamond_2 a)$), we must recursively call **Π-Entail** on the pair of formulae behind the $\Box_1$ operators. We have **Π-Entail**(d, $d \vee \Diamond_2 a$)=**yes** since $\{d\} \subseteq \{d\}$, and hence **Π-Entail**($\Box_1 d$, $\Box_1 (d \vee \Diamond_2 a)$)=**yes**. We have thus shown that this pair of conjuncts satisfies all three conditions, so we set $MatchFound =$ **yes**.

As we have found a match for the first conjunct of ψ, we will move on to the second conjunct which is $\neg c \vee \Diamond_1 (a \wedge b)$. We reset $MatchFound$ to *no*, and we then consider the first conjunct of φ, which fails condition (a) since $\{a\} \not\subseteq \{\neg c\}$. The second conjunct of φ also fails condition (a) since $\{\neg a\} \not\subseteq \{\neg c\}$. The third conjunct fails condition (b) since $\neg c \vee \Diamond_1 (a \wedge b)$ has no $\Diamond_2$-disjuncts. The fourth conjunct fails condition (c) since $\neg c \vee \Diamond_1 (a \wedge b)$ has no $\Box$-formulae as disjuncts..

Finally, to decide whether the fifth conjunct of φ constitutes a match, we call Π-**Entail**$(a \wedge \neg c \wedge \Box_2 \bot,\ a \wedge b)$. We find a match for the first conjunct a of $a \wedge b$. We do not however find a match for the second conjunct b since the conjuncts a and $\neg c$ both falsify condition (a) and the conjunct $\Box_2 \bot$ has no propositional part. Thus, Π-**Entail**$(a \wedge \neg c \wedge \Box_2 \bot,\ a \wedge b)$=**no**, which means that the fifth conjunct of φ does not satisfy condition (c) and $MatchFound$ will still be *no* at the end of the for-loop. It follows that Π-**Entail** will return **no** for the pair of formulae $(\varphi,\ \psi)$, which is the correct answer since $\varphi \not\models \psi$.

We now prove that Π-**Entail** behaves as desired when the input formulae are in prime implicate normal form.

Lemma 6.3.2.
*If φ_1 and φ_2 are both in prime implicate normal form, then the algorithm Π-**Entail** outputs **yes** on input $(\varphi_1,\ \varphi_2)$ if $\varphi_1 \models \varphi_2$.*

Proof. The proof is by induction on $min(\delta(\varphi_1), \delta(\varphi_2))$. We begin with the base case where $\varphi_1 \models \varphi_2$ and $min(\delta(\varphi_1), \delta(\varphi_2)) = 0$, i.e. where one or both of φ_1 and φ_2 is propositional. There are three possibilities: either $\varphi_1 \models \bot$, or $\models \varphi_2$, or neither $\varphi_1 \models \bot$ nor $\models \varphi_2$. In the first case, φ_1 must be $\bot$ (otherwise φ_1 would not be in prime implicate normal form), so the algorithm will return **yes** in Step 1. Similarly, in the second case, we must have $\varphi_2 = \top$, so the algorithm returns **yes** in the first step.

Let us then concentrate on the third case in which $\varphi_1 \not\models \bot$ and $\not\models \varphi_2$. Since $\varphi_1 \models \varphi_2$, it follows that we must also have $\varphi_2 \not\models \bot$ and $\top \not\models \varphi_1$. This means that the conditions for Steps 1 and 2 of Π-**Entail** are not satisfied, so we will proceed to Step 3. Now since $\varphi_1 \models \varphi_2$, it must be the case that φ_1 entails every conjunct of φ_2. As the conjuncts of φ_2 are all clausal formulae (since φ_2 is in prime implicate normal form), it follows from Theorem 3.2.8 that every conjunct in φ_2 is entailed by some prime implicate of φ_1. But since φ_1 is in prime implicate normal form, every prime implicate of φ_1 is equivalent to some conjunct of φ_1. This means that for every conjunct λ of φ_2 there must be some conjunct θ of φ_1 such that $\theta \models \lambda$. If φ_1 is propositional, then so are all its conjuncts, so $\theta \models \lambda$ just in the case that $Prop(\theta) \subseteq Prop(\lambda)$ (by Theorem 2.3.3). It follows that when the algorithm considers the conjuncts λ and θ, it will set $MatchFound = yes$. If instead it is φ_2 which is propositional, then λ is also propositional, so every disjunct of θ must be either a propositional literal which belongs to λ or a formula of the form $\Diamond\psi$ where ψ is unsatisfiable (otherwise we would not have $\theta \models \lambda$). But since θ is in prime implicate normal form it cannot have any unsatisfiable disjuncts, so θ must

be composed only of propositional literals which all appear in λ. This means that the algorithm will mark $MatchFound = yes$ when considering the pair of formulae λ and θ. Thus, in either case, we have that for each conjunct λ of φ_2, there is some conjunct θ of φ_1 for which we will mark $MatchFound = yes$, so **Π-Entail** will return **yes**.

We have just shown that **Π-Entail** returns **yes** whenever the input formulae φ_1 and φ_2 are such that $\varphi_1 \models \varphi_2$ and $min(\delta(\varphi_1), \delta(\varphi_2)) = 0$. Now let us suppose that the result holds whenever we have $min(\delta(\varphi_1), \delta(\varphi_2)) \leq k$ and then show that the result still holds when the minimum depth is $k + 1$.

Let φ_1 and φ_2 be formulae in prime implicate normal form such that $\varphi_1 \models \varphi_2$ and $min(\delta(\varphi_1), \delta(\varphi_2)) = k + 1$. As φ_1 and φ_2 both have positive depth, it follows that they can be neither unsatisfiable nor tautologous (since in that case they would be equal to either $\bot$ or $\top$, both of which have depth zero). That means that the algorithm will proceed directly to Step 3. Let λ be some conjunct of φ_2. Now since $\varphi_1 \models \varphi_2$, we must have $\varphi_1 \models \lambda$. As φ_2 is in prime implicate normal form, λ must be a clause, so Theorem 3.2.8 tells us that there is some prime implicate π of φ_1 such that $\pi \models \lambda$. The formula φ_1 is also in prime implicate normal form, so there must be some conjunct θ of φ_1 such that $\pi \equiv \theta$ and hence such that $\theta \models \lambda$. As λ and θ are both clauses, and λ is non-tautologous, by Theorem 2.3.3 we must have:

(a) $Prop(\theta) \subseteq Prop(\lambda)$

(b) If $Diam_i(\theta) \neq \emptyset$, $\bigvee_{\psi \in Diam_i(\theta)} \psi \models \bigvee_{\zeta \in Diam_i(\lambda)} \zeta$ (or just $\bigvee_{\psi \in Diam_i(\theta)} \psi \models \bot$ if $Diam_i(\lambda) = \emptyset$)

(c) If $\psi \in Box_i(\theta)$, then there is some $\epsilon \in Box_i(\lambda)$ such that $\psi \models \epsilon \vee (\bigvee_{\zeta \in Diam_i(\lambda)} \zeta)$
(or $\psi \models \epsilon$ if $Diam_i(\lambda) = \emptyset$)

Statement (a) means that the first condition of the algorithm is satisfied for the pair λ and θ. As for the second condition, let us suppose that θ has at least one $\Diamond_i$-disjunct. As φ_1 is in prime implicate normal form, there must be exactly one element in $Diam_i(\theta)$, and this element must be satisfiable (otherwise θ would contain an unnecessary disjunct). Let ψ be this formula. Now because of (b) and the fact ψ is satisfiable, $Diam_i(\lambda)$ must be non-empty and ψ must entail the disjunction of the elements in $Diam_i(\lambda)$. But φ_2 is also in prime implicate normal form, so there must be a single element in $Diam_i(\lambda)$, call it ψ'. We thus have $\psi \models \psi'$. Because φ_1 and φ_2 are formulae in prime implicate normal form with $min(\delta(\varphi_1), \delta(\varphi_2)) = k+1$, it follows that ψ and ψ' are also in prime implicate normal form and $min(\delta(\psi), \delta(\psi')) \leq k$. This means the induction hypothesis applies, so **Π-Entail**(ψ, ψ')=**yes**, and hence the second condition of the algorithm is satisfied for the pair λ and θ. Finally, we remark that because of statement (c) above and

condition 3(c)iv of Definition 6.2.1 (which applies to λ and θ since we have assumed φ_1 and φ_2 are in prime implicate normal form) it follows that for each disjunct $\Box_i\psi$ of θ there is some disjunct $\Box_i\psi'$ of λ such that $\psi \models \psi'$. Now ψ and ψ' are formulae in prime implicate normal form (by part 3(c)iii of Definition 6.2.1) such that $min(\delta(\psi), \delta(\psi')) \leq k$ and $\psi \models \psi'$, so according to the induction hypothesis, it must be the case that Π-**Entail**(ψ, ψ')=**yes**. This means that λ and θ satisfy the third and final condition of the algorithm. We have thus shown that for every conjunct λ of φ_2 there is some conjunct θ of φ_1 such that the three conditions of Step 3 are satisfied. This means that the algorithm will return **yes** on input (φ_1, φ_2). □

Lemma 6.3.3.
If φ_1 and φ_2 are both in prime implicate normal form, then the algorithm Π-**Entail** *outputs* ***no*** *on input (φ_1, φ_2) if $\varphi_1 \not\models \varphi_2$.*

Proof. The proof is by induction on $min(\delta(\varphi_1), \delta(\varphi_2))$. We begin with the base case where $\varphi_1 \not\models \varphi_2$ and $min(\delta(\varphi_1), \delta(\varphi_2)) = 0$, i.e. where one or both of φ_1 and φ_2 is propositional. If $\models \varphi_1$ and $\not\models \varphi_2$, then $\varphi_1 = \top$ and $\varphi_2 \neq \top$ (since φ_1 and φ_2 are assumed to be in prime implicate normal form), so the algorithm will return **no** in the second step. Likewise, if $\varphi_2 \models \bot$ and $\varphi_1 \not\models \bot$, then we must have $\varphi_1 \neq \bot$ and $\varphi_2 = \bot$, so the algorithm returns **no** in Step 2. If neither of these cases holds, then φ_1 and φ_2 must both be satisfiable and non-tautologous, and the algorithm proceeds to Step 3. As $\varphi_1 \not\models \varphi_2$, it must be the case that there is some conjunct λ of φ_2 such that $\theta \not\models \lambda$ for every conjunct θ of φ_1. If it is φ_1 that is propositional, then it follows from Theorem 2.3.3 that $Prop(\theta) \not\subseteq Prop(\lambda)$ for every conjunct θ of φ_1. If it is φ_2 that is propositional, then for each conjunct θ of φ_1 either $Prop(\theta) \not\subseteq Prop(\lambda)$ or θ contains modal disjuncts. In either case, we find that each conjunct θ of φ_1 violates at least one of the conditions in Step 3. This means that the algorithm does not set $MatchFound = yes$ at any point when examining the conjunct λ and hence returns **no**.

We have thus shown that Π-**Entail** returns **no** whenever φ_1 and φ_2 are formulae in prime implicate normal form such that $\varphi_1 \not\models \varphi_2$ and $min(\delta(\varphi_1), \delta(\varphi_2)) = 0$. We will now suppose that the same statement holds whenever $min(\delta(\varphi_1), \delta(\varphi_2)) \leq k$ and will show that the result remains true when the minimal depth is $k + 1$.

Let φ_1 and φ_2 be formulae in prime implicate normal form such that $\varphi_1 \not\models \varphi_2$ and $min(\delta(\varphi_1), \delta(\varphi_2)) = k+1$. Since φ_1 and φ_2 are in prime implicate normal form and have positive depth, φ_1 and φ_2 cannot be equal to $\top$ or $\bot$, so the algorithm proceeds directly to Step 3. As $\varphi_1 \not\models \varphi_2$, there must be some conjunct λ of φ_2 such that $\theta \not\models \lambda$ for every conjunct θ of φ_1. According to Theorem 2.3.3, this means

that for every conjunct θ of φ_1 we have one of the following:

(a) $Prop(\theta) \not\subseteq Prop(\lambda)$

(b) For some $1 \leq i \leq n$: $Diam_i(\theta) \neq \emptyset$ and either $Diam_i(\lambda) = \emptyset$ and $\bigvee_{\psi \in Diam_i(\theta)} \psi \not\models \bot$ or $Diam_i(\lambda) \neq \emptyset$ and $\bigvee_{\psi \in Diam_i(\theta)} \psi \not\models \bigvee_{\zeta \in Diam_i(\lambda)} \zeta$

(c) For some $1 \leq i \leq n$ and $\psi \in Box_i(\theta)$, there is no $\epsilon \in Box_i(\lambda)$ such that $\psi \models \epsilon \vee (\bigvee_{\zeta \in Diam_i(\lambda)} \zeta)$ (or $\psi \models \epsilon$ if $Diam_i(\lambda) = \emptyset$)

If (a) holds, then the first condition of Step 3 is violated. If (b) holds, then either $Diam_i(\lambda) = \emptyset$ or $\psi \not\models \psi'$, where $\psi \in Diam_i(\theta)$ and $\psi' \in Diam_i(\lambda)$ (remember that since φ_1 and φ_2 are in prime implicate normal form, the clauses λ and θ can have at most one $\Diamond_i$-disjunct each). In the first case, the second condition of Step 3 is violated since $Diam_i(\lambda)$ is empty. In the second case, the condition is also violated since ψ and ψ' are formulae in prime implicate normal form such that $\psi \not\models \psi'$ and $min(\delta(\psi), \delta(\psi')) \leq k$, so according to the induction hypothesis Π-**Entail**(ψ, ψ')=**no**. Finally, if (c) holds, then for some disjunct $\Box_i\psi$ of θ and every disjunct $\Box_i\psi'$ of λ we have $\psi \not\models \psi' \vee \zeta$ where $\zeta \in Diam_i(\lambda)$ (or simply $\psi \not\models \psi'$ if $Diam_i(\lambda)$ is empty). But since λ is in prime implicate normal form, if $\zeta \in Diam_i(\lambda)$ then $\psi' \equiv \psi' \vee \zeta$. So we get that $\psi \not\models \psi'$, and hence by the induction hypothesis (which applies since ψ and ψ' are in prime implicate normal form and $min(\delta(\psi), \delta(\psi')) = k$) that Π-**Entail**(ψ,ψ') returns **no**. We have thus shown that for every conjunct θ of φ_1 at least one of the three conditions of Step 3 will not be satisfied for the pair λ and θ. This means that when the algorithm has finished its examination of the conjunct λ, the variable $MatchFound$ will still be set to *no*, so Π-**Entail** will return **no**. □

Lemma 6.3.4.
The algorithm Π-**Entail** *terminates in linear time in* $|\varphi_1||\varphi_2|$ *(hence at most quadratic time in* $|\varphi_1| + |\varphi_2|$*) when given formulae* φ_1 *and* φ_2 *as input.*

Proof. The algorithm Π-**Entail** compares at most once each pair of symbols from φ_1 and φ_2, and the comparison takes constant time, yielding an overall complexity which is linear in $|\varphi_1||\varphi_2|$. □

Theorem 6.3.5.
Entailment of formulae in prime implicate normal form can be decided in quadratic time in the size of the input.

Proof. Direct consequence of Lemmas 6.3.2, 6.3.3, and 6.3.4. □

Corollary 6.3.6.
Equivalence of formulae in prime implicate normal form can be decided in quadratic time in the size of the input.

If we examine the proofs of Lemmas 6.3.2 and 6.3.3, we remark that only some of the properties of prime implicate normal form are used for the first formula and others for the second formula. It is thus interesting to investigate exactly what properties are needed to ensure the correctness of our structural comparison algorithm. In particular, it would be nice to loosen the conditions on the second formula, as this would allow us to more easily pose entailment queries to formulae compiled into prime implicate normal form. In the following theorem, we make explicit the conditions that must be placed on the two input formulae in order to ensure the successful functioning of Π-**Entail**. For the statement of the theorem, we require the following definition:

Definition 6.3.7.
A formula φ is said to be in *extended conjunctive normal form* if and only if φ is a conjunction of clauses λ_i, and for every ψ such that $\Diamond_k\psi$ or $\Box_k\psi$ is a disjunct of some λ_i, ψ is in extended conjunctive normal form.

Example 6.3.8.
The formula $(a \vee b) \wedge \Diamond_2(a \wedge (\neg b \vee \Box_1 c))$ is in extended conjunctive normal form, but $a \wedge \Diamond_2((a \wedge b) \vee c)$ is not since $(a \wedge b) \vee c$ is not a conjunction of clauses.

Theorem 6.3.9.
Let φ_1 be a formula from $\mathcal{K}_n$ in extended conjunctive normal form such that:

- *every prime implicate of φ_1 is equivalent to some conjunct of φ_1*
- *every formula ψ such that $\Diamond_k\psi$ or $\Box_k\psi$ is a subformula of φ_1 is such that every prime implicate of ψ is equivalent to some conjunct of ψ*
- *if ψ is an unsatisfiable subformula of φ_1, then $\psi = \bot$*
- *no satisfiable subclause of φ_1 contains an unsatisfiable disjunct*[3]

Let φ_2 be a formula from $\mathcal{K}_n$ in extended conjunctive normal form such that:

- *every clausal subformula λ of φ_2 is such that*
  - *$|Diam_k(\lambda)| \leq 1$ for all $1 \leq k \leq n$*
  - *if $\gamma \in Diam_k(\lambda)$ and $\zeta \in Box_k(\lambda)$ for some $1 \leq k \leq n$, then $\gamma \models \zeta$*
- *if ψ is a tautologous subformula of φ_2, then $\psi = \top$*
- *no non-tautologous conjunction appearing in φ_2 contains a tautologous conjunct*[4]

3. Any formula φ_1 which satisfies the third bullet can be easily transformed into an equivalent formula satisfying the fourth bullet: we simply remove any disjuncts $\bot$ from the clauses appearing in φ_1. Alternatively, we can modify the algorithm Π-**Entail** to allow for the case where disjuncts in the first formula may be $\bot$.

4. If the previous bullet is satisfied by φ_2, then we can simply remove any conjuncts $\top$ from the conjunctions in φ_2. Or we could slightly modify Π-**Entail** so as to allow the second formula to have conjuncts of the form $\top$.

Then the algorithm Π-**Entail** *outputs* ***yes*** *on input* (φ_1, φ_2) *if and only if* $\varphi_1 \models \varphi_2$.

Proof. Lemmas 6.3.2 and 6.3.3 are straightforwardly modified to handle input formulae of the types described in the statement of the theorem in place of formulae in prime implicate normal form. □

Note 6.3.10.
In what follows, we will need to make reference several times to the conditions on φ_1 and φ_2 outlined in Theorem 6.3.9. For this reason, and to simplify the presentation, we will call the conditions placed on φ_1 the *conditions on left-hand-side formulae*, and the conditions on φ_2 the *conditions on right-hand-side formulae*.

Theorem 6.3.9 is important since it allows us to use our structural comparison algorithm on a wider class of queries. The following three results illustrate some specific types of queries that are made possible by this theorem.

Theorem 6.3.11.
Let φ_1 *be a formula in prime implicate normal form, and let* φ_2 *be a term with respect to definition* ***D1****. Then it can be decided in quadratic time in* $|\varphi_1| + |\varphi_2|$ *whether* $\varphi_1 \models \varphi_2$.

Proof. Let φ_1 and φ_2 be as stated, and consider the following procedure:

Step 1 If $\Box_k\psi$ is a subformula of φ_2 such that ψ is built up uniquely from $\top$, $\wedge$, and universal modalities $\Box_i$ $(1 \leq i \leq k)$, then replace all occurrences of $\Box_k\psi$ by $\top$ in φ_2. Repeat Step 1 until no such subformulae remain.

Step 2 If ψ is a subterm of φ_2 with only conjuncts $\top$, replace ψ with $\top$. Otherwise, if ψ is a subterm of φ_2 having some (but not all) conjuncts equal to $\top$, then remove all conjuncts $\top$ from ψ. Repeat Step 2 until no such subterms remain.

We note that any formula constructed uniquely out of $\top$, conjunction, and the universal modal operators must be a tautology. This means that the modifications in the above procedure are equivalence-preserving, so the formula we obtain, call it φ_2', is equivalent to φ_2. It follows then that $\varphi_1 \models \varphi_2$ if and only if $\varphi_1 \models \varphi_2'$.

We claim that φ_2' satisfies the requirements of right-hand-side formulae. It is easy to see that φ_2' is in extended conjunctive normal form. It also easy to see that it satisfies the next two bullets of Theorem 6.3.9, since it does not contain any disjunctions, making it impossible to have more than one $\Diamond_k$-disjunct, or pair of disjuncts $\Diamond_k\psi$ and $\Box_k\gamma$, in a clausal subformula. Finally, because of the modifications we have made, there can be no tautologous subformula in φ_2' which is not equal to $\top$ nor any subformula which is a non-tautologous conjunction having a conjunct $\top$

All of the conditions outlined in Theorem 6.3.9 for right-hand-side formulae are satisfied by φ_2', and φ_1 clearly satisfies the requirements on left-hand side formulae, being a formula in prime implicate normal form. It follows that we can use **Π-Entail** to decide whether $\varphi_1 \models \varphi_2'$.

Then to complete the proof, we just need to show that the above procedure for transforming φ_2 into φ_2' runs in quadratic time in $|\varphi_2|$. This is quite easy to see: the first step can only be repeated at most $|\varphi_2|$ times since we remove a $\Box$-formula at each iteration (and never add any $\Box$-formulae), and there cannot be more than $|\varphi_2|$ $\Box$-formulae in φ_2. Moreover, each iteration takes only linear time in $|\varphi_2|$ since we simply scan the symbols in one of the subformulae of φ_2, and the modifications made to φ_2 in Step 1 never increase its length. Step 2 can also be carried out in linear time, since it involves a single pass through φ_2, and the modified $|\varphi_2|$ has equal or smaller length to the original formula. Thus, the modification of φ_2 into φ_2' takes only quadratic time in $|\varphi_2|$. As $|\varphi_2'| \leq |\varphi_2|$, and it is possible to decide $\varphi_1 \models \varphi_2'$ in time quadratic in $|\varphi_1| + |\varphi_2'|$ using **Π-Entail** (Lemma 6.3.4), we obtain a method for deciding $\varphi_1 \models \varphi_2$ in time quadratic in $|\varphi_1| + |\varphi_2|$. □

Example 6.3.12.
Let φ be the formula in prime implicate normal form from Example 6.2.5:

$$\begin{aligned}
& (\, a \vee \Box_2 \Box_1 d \,) \\
\wedge \;& (\, \neg a \vee \Diamond_2 (\, a \wedge b \,) \,) \\
\wedge \;& (\, \Box_2 (\, (\, \Box_1 d \vee a \,) \wedge (\, \Box_1 d \vee b \,) \,) \vee \Diamond_2 (\, a \wedge b \,) \,) \\
\wedge \;& \Box_1 (\, (\, a \vee \Diamond_2 \top \,) \wedge (\, \Box_2 \bot \vee c \,) \wedge (\, a \vee c \,) \,) \\
\wedge \;& \Diamond_1 (\, a \wedge \neg c \wedge \Box_2 \bot \,)
\end{aligned}$$

and let τ be the following **D1**-term

$$\Diamond_1 (\, a \wedge \Box_2 (\, b \wedge c \,) \,) \wedge \Box_3 (\, \top \wedge \Box_1 \top \,)$$

We want to test whether $\varphi \models \tau$. To do this, we first simplify τ using the procedure given in the proof of Theorem 6.3.11. This involves replacing the tautologous subformula $\Box_3 (\, \top \wedge \Box_1 \top \,)$ with $\top$, and then removing $\top$ from the conjunction. We obtain the equivalent **D1**-term τ':

$$\Diamond_1 (\, a \wedge \Box_2 (\, b \wedge c \,) \,)$$

We can now use **Π-Entail** to decide whether $\varphi \models \tau'$, and hence whether $\varphi \models \tau$. As neither φ nor τ' is equal to $\top$ or $\bot$, we proceed directly to Step 3, in which we compare the sole conjunct of τ to each of the conjuncts of φ to τ' in order to find

some conjunct which satisfies the three requirements. The first two conjuncts of φ do not satisfy requirement (a) since they possess propositional disjuncts which do not appear in τ'. The third conjunct falsifies requirements (b) and (c) as its $\Box_2$- and $\Diamond_2$-disjuncts do not match up to the unique disjunct in τ'. The fourth conjunct of φ does not satisfy the requirements either, as it possesses a $\Box$-disjunct, and τ' does not. Finally, the fifth conjunct of φ trivially satisfies requirements (a) and (c) since it does not have any propositional or $\Box$-disjuncts. In order to show that this conjunct also satisfies requirement (b), we need to call **Π-Entail** on the pair of formulae $a \wedge \neg c \wedge \Box_2 \bot$ and $a \wedge \Box_2 (b \wedge c)$. The algorithm will return **yes** on this input, since the conjunct a in the second formula matches up with the conjunct a in the first formula, and the conjunct $\Box_2 (b \wedge c)$ of the second formula matches with the conjunct $\Box_2 \bot$ (since **Π-Entail** always returns **yes** when the first formula is $\bot$). Thus, while examining the fifth conjunct of φ, we will set $MatchFound = yes$, and hence **Π-Entail** will return **yes** at the end of Step 3.

Theorem 6.3.13.
Let φ_1 be a formula in prime implicate normal form, and let φ_2 be a formula in extended conjunctive normal form such that for every clausal subformula λ of φ_2 and every $1 \leq k \leq n$ either $|Diam_k(\lambda)| = 0$ or $|Diam_k(\lambda)| = 1$ and $|Box_k(\lambda)| = 0$. Then it can be decided in quadratic time in $|\varphi_1| + |\varphi_2|$ whether $\varphi_1 \models \varphi_2$.

Proof. Let φ_1 and φ_2 be as described. In order to be able to apply Theorem 6.3.9, we need to show how to transform φ_2 into an equivalent formula satisfying all of the requirements for right-hand-side formulae. Let us consider the following recursive procedure **RemoveTaut** which takes as input a formula $\psi = \lambda_1 \wedge ... \wedge \lambda_l$ in extended conjunctive normal form:

Step 1 For each λ_i: Replace each disjunct $\Box_k \gamma$ (resp. $\Diamond_k \gamma$) by $\Box_k$**RemoveTaut**(γ) (resp. $\Diamond_k$**RemoveTaut**(γ)). Afterwards, check whether λ_i contains complementary propositional disjuncts or some disjunct of the form $\Box_k \top$, and replace λ_i by $\top$ if this is the case.

Step 2 If all conjuncts of ψ are $\top$, replace ψ by $\top$, else remove all conjuncts $\top$ from ψ.

Step 3 Return the modified formula ψ.

It is easy to see that this procedure outputs a formula which is equivalent to the input formula, as each of the modifications is equivalence-preserving. We claim furthermore that if the input is a formula in extended conjunctive normal form such that for every clausal subformula λ and every $1 \leq k \leq n$ either $|Diam_k(\lambda)| = 0$ or $|Diam_k(\lambda)| = 1$ and $|Box_k(\lambda)| = 0$, then the output formula satisfies all conditions

of the right-hand side formula. The proof is by induction on the depth of the input formula. The base case is when $\delta(\psi) = 0$. In this case, in Step 1, we replace all tautologous clauses by $\top$, and in Step 2, we remove extra $\top$ conjuncts from ψ. The output formula is thus either $\top$ or a conjunction of non-tautologous propositional clauses, so all conditions of right-hand-side formulae are satisfied.

Let us next assume that our statement holds whenever the input formula has depth at most m. Let ψ be a formula in extended conjunctive normal form of depth $m+1$ such that for every clausal subformula λ and every $1 \leq k \leq n$ either $|Diam_k(\lambda)| = 0$ or $|Diam_k(\lambda)| = 1$ and $|Box_k(\lambda)| = 0$, and let ψ' be the output of the above procedure on input ψ. As we never add $\Box$- or $\Diamond$-formulae during the procedure, we can be sure that ψ' is also such that for every clausal subformula λ and every $1 \leq k \leq n$ either $|Diam_k(\lambda)| = 0$ or $|Diam_k(\lambda)| = 1$ and $|Box_k(\lambda)| = 0$. It remains to be shown that every tautologous subformula of ψ' is equal to $\top$. Suppose that this is not the case. Then there must be some subformula ζ such that $\models \zeta$ but $\zeta \neq \top$. Suppose first that ζ appears in the scope of one or more modal operators. Then that means that there is some disjunct $\Diamond_k\gamma$ or $\Box_k\gamma$ of one of the clausal conjuncts of ψ', such that ζ is a subformula of γ. We know from the definition of **RemoveTaut** that there must be some subformula σ of ψ such that $\gamma = \mathbf{RemoveTaut}(\sigma)$. As σ appears in ψ behind a modal operator, it must be a formula in extended conjunctive normal form of depth at most m such that every clausal subformulae λ of σ is such that either $|Diam_k(\lambda)| = 0$ or $|Diam_k(\lambda)| = 1$ and $|Box_k(\lambda)| = 0$ for $1 \leq k \leq n$. This means that the formula $\gamma = \mathbf{RemoveTaut}(\sigma)$ satisfies all conditions of Theorem 6.3.9. In particular, γ cannot contain ζ as a subformula. It follows then that ζ must be a subformula of ψ' which appears outside the modal operators. If ζ is a literal, then it must be of the form $\Box_k\mu$, since a propositional literal or $\Diamond$-formula cannot be a tautology. Since $\Box_k\mu$ is a tautology, we must have $\mu \equiv \top$. But we have just shown that all tautologous subformulae appearing behind the modal operators in ψ' are equal to $\top$, so $\mu = \top$. But this is a contradiction, since we would have replaced the clause containing $\Box_k\top$ with $\top$ in Step 1 (and then removed the clause in Step 2). Suppose next that ζ is a clausal conjunct of ψ'. Then by Theorem 2.3.1 it must contain either a pair of complementary propositional literals or a tautologous $\Box$-disjunct. In the latter case, we know from preceding discussion that the tautologous $\Box$-disjunct would have been turned into a formula of form $\Box_k\top$ in Step 1. In either case, we would have replaced the conjunct ζ by $\top$ in Step 1 and deleted it in Step 2, contradicting the fact that ζ is a conjunct of ψ'. The only remaining possibility is that ζ is a conjunction of clausal conjuncts of ψ', but this too we can rule out since we have just shown that any tautologous clausal conjunct of ψ' is equal to $\top$, and

we either delete all such conjuncts in Step 2, or replace them by a single conjunct $\top$. We have thus shown that all of the tautologous subformulae of ψ' are equal to $\top$. This means in particular that there can be no conjunctions in ψ' which have tautologous conjuncts but are not themselves tautologies, since they would have a conjunct $\top$, and we have removed in Step 2 all $\top$ conjuncts from ψ.

We have just shown how to transform the formula φ_2 into an equivalent formula φ_2' which satisfies all of the requirements of right-hand-side formulae of Theorem 6.3.9. This means that we can test whether $\varphi_1 \models \varphi_2$ by using **Π-Entail** to decide whether $\varphi_1 \models \varphi_2'$. According to Lemma 6.3.4, the algorithm **Π-Entail** will take at most quadratic time in $|\varphi_1| + |\varphi_2|$ since $|\varphi_2'| \leq |\varphi_2|$ (all of the modifications in the above procedure decrease the size of the input formula). Moreover, the transformation of φ_2 into an equivalent formula φ_2' clearly takes at most quadratic time in $|\varphi_2|$, which means that it can be decided in time quadratic in $|\varphi_1| + |\varphi_2|$ whether $\varphi_1 \models \varphi_2$. □

Example 6.3.14.
Let φ be defined as in Example 6.2.5:

$$\begin{array}{ll} & (\, a \vee \Box_2 \Box_1 d \,) \\ \wedge & (\, \neg a \vee \Diamond_2 (\, a \wedge b \,) \,) \\ \wedge & (\, \Box_2 (\, (\, \Box_1 d \vee a \,) \wedge (\, \Box_1 d \vee b \,) \,) \vee \Diamond_2 (\, a \wedge b \,) \,) \\ \wedge & \Box_1 (\, (\, a \vee \Diamond_2 \top \,) \wedge (\, \Box_2 \bot \vee c \,) \wedge (\, a \vee c \,) \,) \\ \wedge & \Diamond_1 (\, a \wedge \neg c \wedge \Box_2 \bot \,) \end{array}$$

and let ψ be the formula

$$(\, b \vee c \vee \Diamond_1 \Box_2 \Diamond_1 \neg a \,) \wedge (\, a \vee \Box_2 (\, \neg a \vee \Box_1 (\, d \vee \neg d \,)\,)\,)$$

We know from Example 6.2.5 that φ is in prime implicate normal form, and it can be verified that ψ is a formula in extended conjunctive normal form satisfying the conditions of Theorem 6.3.13. Applying the transformation from the proof of Theorem 6.3.13 to ψ yields the equivalent formula

$$b \vee c \vee \Diamond_1 \Box_2 \Diamond_1 \neg a$$

since **RemoveTaut**$(d \vee \neg d)$=$\top$, and hence **RemoveTaut**$(\neg a \vee \Box_1 (\, d \vee \neg d \,))$=$\top$. Now that we have put ψ in the proper form, we can use **Π-Entail** to test whether $\varphi \models \psi$. As transformed ψ is comprised of a single clause, we just need to check whether one of the clausal conjuncts of φ satisfies the three conditions with regards to ψ. The first two conjuncts fail condition (a) since their propositional disjuncts

do not belong to $\{b, c\}$. The third and fourth conjuncts of φ fail condition (c) since ψ does not have any $\Box$-formulae as disjuncts. The fifth conjunct satisfies conditions (a) and (c) by default since it does not have any propositional or $\Box$-disjuncts. To determine whether this conjunct also satisfies condition (b), we need to call **Π-Entail** on the pair of formulae $(a \wedge \neg c \wedge \Box_2 \bot, \Box_2 \Diamond_1 a)$. This call will succeed since **Π-Entail**$(\bot, \Diamond_1 a)$=**yes**. It follows that $MatchFound$ will be set to *yes* upon examination of the fifth conjunct, and so **Π-Entail** will return **yes**, as desired.

Theorem 6.3.15.
Let φ_1 be a formula in prime implicate normal form, and let φ_2 be a formula in extended conjunctive normal form of depth 1 such that for every subformula $\Diamond_k \psi$ or $\Box_k \psi$, the formula ψ is a clause. Then it can be decided in polynomial time in $|\varphi_1| + |\varphi_2|$ whether $\varphi_1 \models \varphi_2$.

Proof. Let φ_1 and φ_2 be as described. Consider the following procedure:

Step 1 Apply the following modifications to φ_2:

(a) For each conjunct λ and index k: if $Diam_k(\lambda) = \{\psi_1, ..., \psi_m\}$ where $m > 1$, replace λ by $\lambda \setminus \{\Diamond_k \psi_1, ..., \Diamond_k \psi_m\} \cup \{\Diamond_k \top\}$ if $\psi_1 \vee ... \vee \psi_m$ is a tautologous propositional clause and by $\lambda \setminus \{\Diamond_k \psi_1, ..., \Diamond_k \psi_m\} \cup \{\Diamond_k (\psi_1 \vee ... \vee \psi_m\}$ otherwise.

(b) For each conjunct λ and index k: if $Diam_k(\lambda) = \{\psi\}$ and $Box_k(\lambda) = \{\gamma_1, ..., \gamma_p\}$, replace λ by $\lambda \setminus \{\Box_k \gamma_1, ..., \Box_k \gamma_p\} \cup \{\Box_k (\gamma_1 \vee \psi), ..., \Box_k (\gamma_p \vee \psi)\}$.

(c) If $Prop(\lambda)$ contains two complementary atomic literal formulae, or if there is some disjunct $\Box_k \zeta$ where ζ is a tautologous propositional clause, remove λ from φ_2.

Step 2 If all conjuncts of φ_2 have been removed, return $\top$. Otherwise, return the modified φ_2.

We claim that the formula returned by this procedure, call it φ_2', is equivalent to the original formula φ_2, which means that we can test $\varphi_1 \models \varphi_2$ by testing $\varphi_1 \models \varphi_2'$. We claim furthermore that φ_2' satisfies all of the conditions of right-hand-side formulae outlined in Theorem 6.3.9, which means that according to Theorem 6.3.9 and Lemma 6.3.4, it is possible to test whether $\varphi_1 \models \varphi_2'$ in quadratic time in $|\varphi_1| + |\varphi_2'|$. This is sufficient to show the result since clearly the above transformation operates in polynomial time (hence space) in the length of the input formula φ_2.

Showing that φ_2' is equivalent to φ_2 is straightforward. All of the transformations in Step 1 are equivalence-preserving: part (a) is equivalence-preserving because of item 5 of Theorem 2.3.1; part (b) is equivalence-preserving because of item 9 of Theorem 2.3.1; part (c) is equivalence-preserving since any clause with

complementary propositional literals or with a tautologous $\Box$-disjunct must be tautologous. Finally, Step 2 is equivalence-preserving since if all conjuncts of φ_2 were removed in Step 1, then all of φ_2's conjuncts are tautologies, so $\varphi_2 \equiv \top$.

We now show that the formula φ_2' satisfies the requirements of right-hand-side formulae. We first note that φ_2' is in extended conjunctive normal form, being a conjunction of clauses such that the formulae behind the modalities are all propositional clauses. We then note that because of Step 1(a) of the above transformation, there can be at most one $\Diamond_k$-disjunct in each clause appearing in φ_2'. Also, because of Step 1(b), we know that if λ is a clausal subformula of φ_2' and $\gamma \in Diam_k(\lambda)$ and $\zeta \in Box_k(\lambda)$ for some $1 \leq k \leq n$, then $\gamma \models \zeta$. Because of Step 1(a) and 1(c), we know that the only possible tautologous subformulae of φ_2' are φ_2' itself or some formula appearing behind a $\Diamond_k$ modality. In either case, the tautologous subformula must be equal to $\top$. $\square$

Example 6.3.16.
Let φ be defined as in Example 6.2.5:

$$\begin{array}{ll} & (\, a \vee \Box_2 \Box_1 d \,) \\ \wedge & (\, \neg a \vee \Diamond_2 (\, a \wedge b \,) \,) \\ \wedge & (\, \Box_2 (\, (\, \Box_1 d \vee a \,) \wedge (\, \Box_1 d \vee b \,) \,) \vee \Diamond_2 (\, a \wedge b \,) \,) \\ \wedge & \Box_1 (\, (\, a \vee \Diamond_2 \top \,) \wedge (\, \Box_2 \bot \vee c \,) \wedge (\, a \vee c \,) \,) \\ \wedge & \Diamond_1 (\, a \wedge \neg c \wedge \Box_2 \bot \,) \end{array}$$

and let ψ be the formula

$$(\, \neg c \vee \Diamond_2 b \vee \Box_2 \neg b \,) \wedge (\, \neg a \vee \neg b \vee \Diamond_2 a \vee \Diamond_2 b \,)$$

We know from Example 6.2.5 that φ is in prime implicate normal form, and it can be verified that ψ is a formula in extended conjunctive normal form satisfying the conditions of Theorem 6.3.15. Let us begin then by applying the transformation in the proof of Theorem 6.3.15 to ψ. In Step 1(a) of the transformation, we replace the disjuncts $\Diamond_2 a$ and $\Diamond_2 b$ in the second conjunct by a single disjunct $\Diamond_2 (\, a \vee b \,)$. Then in Step 1 (b) we replace the disjunct $\Box_2 \neg b$ in the first conjunct by $\Box_2 (\, \neg b \vee b \,)$. This means that in Step 1(c), we will remove the first conjunct from ψ. Thus, at the end of the transformation, we have $\psi = \neg a \vee \neg b \vee \Diamond_2 (\, a \vee b \,)$. We can then call **$\Pi$-Entail** to decide whether $\varphi \models \psi$. The algorithm will output **yes** since the second conjunct of φ satisfies all three conditions with respect to the unique conjunct of ψ.

Remark 6.3.17.
We cannot extend the previous result to the entire class of $\mathcal{K}_n$ formulae in NNF which are conjunctions of **D1**-clauses. This is because, as we saw earlier in the proof of Lemma 3.1.4, deciding whether a **D1**-term is unsatisfiable is an NP-complete task, which means that the dual problem of deciding whether a **D1**-clause is a tautology must also be NP-complete. Similarly, we can show that deciding whether a **D2**-clause is a tautology is also NP-complete. This means that *there cannot exist any compilation method for $\mathcal{K}_n$ formulae that allows one to tractably answer all **D2**-clause entailment queries.*

In the previous theorems, we exhibited some specific tractable classes of queries for formulae in prime implicate normal form. We now consider the problem of posing arbitrary entailment queries to formulae in prime implicate normal form. We recall that in the case of propositional logic, one can test whether an arbitrary formula is entailed by a formula in prime implicate normal form by first putting the formula in conjunctive normal form and then using structural comparison to decide entailment. The transformation to conjunctive normal form involves a single-exponential blowup in formula size in the worst-case. We show an analogous result for $\mathcal{K}_n$, namely that every formula in $\mathcal{K}_n$ can be transformed into an equivalent formula which satisfies the conditions for right-hand-side formulae and which is no more than single-exponentially larger. This means that we can test whether an arbitrary formula ψ is a logical consequence of a formula φ in prime implicate normal form by first making ψ satisfy the right-hand-side conditions and then running the algorithm **Π-Entail**.

Theorem 6.3.18.
There exists a polynomial function f such that for every formula φ there exists an equivalent formula φ' which satisfies all the conditions for right-hand-side formulae and is such that $|\varphi'| \leq 2^{f(|\varphi|)}$.

Proof. We assume without loss of generality that formulae are in NNF. Define $rhs(\varphi)$ as the length of the shortest formula which is equivalent to φ and satisfies all of the conditions on right-hand-side formulae. We will let $max\text{-}rhs_l(k)$ denote the maximum value of rhs for formulae having depth at most k and at most l distinct literal subformulae.

We would like to place some upper bounds on the value of $max\text{-}rhs_l(k)$. We remark that if φ is tautologous or a contradiction, then $rhs(\varphi) = 1$, since $\top$ and $\bot$ satisfy all right-hand-side conditions. Thus, we can restrict our attention to formulae which are satisfiable and non-tautologous. We begin by considering the case of propositional formulae. We remark that there are only single-exponentially

many non-equivalent propositional clauses on m variables, which means that we can find some polynomial function q such that every propositional formula built using at most m propositional variables is equivalent to some formula in conjunctive normal form with length at most $2^{q(m)}$. We can assume without loss of generality that the CNF formula has no tautologous conjuncts, and hence satisfies all right-hand-side conditions. As the number of propositional variables appearing in a formula can never exceed the number of distinct literal subformulae appearing in the formula, it follows that there exists some polynomial function p such that $max\text{-}rhs_l(0) \leq 2^{p(l)}$.

Now that we have obtained an upper bound on $max\text{-}rhs_l(0)$, we try to obtain an upper bound on $max\text{-}rhs_l(k+1)$ in terms of $max\text{-}rhs_l(k)$. Consider some formula φ with depth $k+1$ and having at most l distinct literal subformulae. We first use the function **Cnf** to rewrite φ as an equivalent conjunction of clauses $\lambda_1 \wedge ... \wedge \lambda_m$. We can assume without loss of generality that the conjuncts λ_1, ..., λ_m are all non-tautologous and mutually non-equivalent and that they contain no redundant disjuncts (otherwise we can simply remove all unnecessary conjuncts and disjuncts, resulting in an even shorter formula). We will now transform each λ_i to make it satisfy the conditions of right-hand-side formulae. First, if there are multiple $\Diamond_j$-disjuncts, we group them into a single $\Diamond_j$ disjunct. Specifically, for each $1 \leq j \leq n$ such that $\Diamond_j(\lambda_i) \geq 2$, if $\Diamond_j(\lambda_i) = \{\psi_1, ..., \psi_r\}$, the we replace the disjuncts $\Diamond_j \psi_1$, ..., $\Diamond_j \psi_r$ by the single disjunct $\Diamond_j (\psi_1 \vee ... \vee \psi_r)$. Secondly, for each disjunct of the form $\Box_j \chi$ such that $\Diamond_j(\lambda_i) = \{\psi\}$ (because of the previous step, we know there to be at most one element in $\Diamond_j(\lambda_i)$), we replace $\Box_j \chi$ by $\Box_j (\chi \vee \psi)$. We remark that these two modifications are equivalence-preserving, so each modified λ_i is equivalent to the original clause λ_i. Notice also that if ζ is such that $\Box_j \zeta$ or $\Diamond_j \zeta$ is a disjunct of the modified λ_i, then ζ must have depth at most $\delta(\varphi) - 1$ and must have at most l distinct literal subformulae. The latter holds since the literal subformulae appearing in the output of **Cnf** on φ are all literal subformulae of φ, and the set of literal subformulae appearing in a disjunction of formulae is equal to the union of the literal subformulae of the disjuncts. We can thus apply the induction hypothesis to all formulae ζ such that $\Box_j \zeta$ or $\Diamond_j \zeta$ is a disjunct of the modified λ_i. Specifically, we find that for each such ζ, there is a formulae ζ' which is equivalent to ζ, has length at most $max\text{-}rhs_l(k)$, and satisfies all right-hand side formulae. Let us then substitute for each formulae ζ the formula ζ'. We remark that the clause resulting from applying the preceding modifications to λ, call it λ'_i, is a clause which is equivalent to λ_i and satisfies all right-hand-side conditions. It follows then that the conjunction $\varphi' = \lambda'_1 \wedge ... \wedge \lambda'_m$ is a formula equivalent to φ which satisfies all right-hand-side conditions.

We now consider the length of φ'. We first remark that there can be at most 2^l

conjuncts in φ' since **Cnf** outputs at most 2^l mutually non-equivalent clauses when the input formula in NNF has at most l mutually non-equivalent literal subformulae (by Theorem 2.4.11). Moreover, we also know from Theorem 2.4.11 that the clauses output by **Cnf** all have at most l mutually non-equivalent disjuncts, so each λ_i has no more than l disjuncts. As the modifications to the λ_i never increase their number of disjuncts, it follows that each modified clause λ_i' has at most l disjuncts. Finally, we know that each disjunct has length at most $max\text{-}rhs_l(k)+1$, since it is either a propositional disjunct, or of the form $\Box_j \zeta'$ or $\Diamond_j \zeta'$, in which case we already showed above that $|\zeta'| \leq max\text{-}rhs_l(k)$. Thus, each conjunct λ_i' can have length at most $l*(max\text{-}rhs_l(k)+1+1)$ (the extra one is for the disjunction symbols between the disjuncts). This means that φ' has length at most $2^l*(l*(max\text{-}rhs_l(k)+2)+1)$ (the extra one is for conjunction symbols between the conjuncts). We thus have

$$max\text{-}rhs_l(k+1) \leq 2^l * (l * (max\text{-}rhs_l(k)+2)+1)$$

From this we can derive that

$$max\text{-}rhs_l(k) \in O((2^l * l)^k * 2^{p(l)})$$

As both the depth of φ and the number of mutually non-equivalent literal subformulae in φ are bounded above by $|\varphi|$, we find that

$$rhs(\varphi) \in O((2^{|\varphi|} * |\varphi|)^k * 2^{p(|\varphi|)})$$

We have thus shown that every formula is equivalent to a formula at most single-exponentially larger which satisfies all right-hand-side conditions. □

6.3.2 Tractable uniform interpolation

As we saw in Chapter 2, the $\mathcal{L}$-interpolant of a formula corresponds to the finest approximation of the formula over a given signature. We show in this subsection that it is easy to generate $\mathcal{L}$-interpolants of formulae in prime implicate normal form.

We introduce an algorithm Π-**LangInt** for computing an $\mathcal{L}$-interpolant of a given formula in prime implicate normal form. The basic idea is to remove all subclauses which have either a propositional disjunct $(\neg)a$ with $a \notin \mathcal{L}$ or a disjunct of the form $\Box_i\psi$ or $\Diamond_i\psi$ where $i \notin \mathcal{L}$.

To illustrate the functioning of Π-**LangInt**, we detail its execution on the formula φ from Example 6.2.5:

Algorithm 6.2 Π**-LangInt**

Input: a formula φ in prime implicate normal form

Output: an $\mathcal{L}$-interpolant of φ

(1) Set $\Pi = \emptyset$.

(2) For each conjunct λ of φ

If $a \in \mathcal{L}$ for every disjunct $(\neg)a$ of λ and $i \in \mathcal{L}$ for every disjunct $\Box_i\psi$ or $\Diamond_i\psi$ of λ, then

(a) Let λ' be the formula obtained from λ by replacing each disjunct $\Box_i\psi$ by $\Box_i\Pi$-**LangInt**$(\psi, \mathcal{L})$ and each disjunct $\Diamond_i\psi$ by $\Diamond_i\Pi$-**LangInt**$(\psi, \mathcal{L})$

(b) If there is no disjunct of λ' of the form $\Box_i\top$, then add λ' to Π

(3) Return the conjunction of the formulae in Π if $\Pi \neq \emptyset$, otherwise return $\top$.

Example 6.3.19.

Let φ be defined as in Example 6.2.5:

$$
\begin{array}{ll}
 & (\, a \vee \Box_2 \Box_1 d \,) \\
\wedge & (\, \neg a \vee \Diamond_2 (\, a \wedge b \,) \,) \\
\wedge & (\, \Box_2 (\, (\, \Box_1 d \vee a \,) \wedge (\, \Box_1 d \vee b \,) \,) \vee \Diamond_2 (\, a \wedge b \,) \,) \\
\wedge & \Box_1 (\, (\, a \vee \Diamond_2 \top \,) \wedge (\, \Box_2 \bot \vee c \,) \wedge (\, a \vee c \,) \,) \\
\wedge & \Diamond_1 (\, a \wedge \neg c \wedge \Box_2 \bot \,)
\end{array}
$$

and let $\mathcal{L} = \{1, 2, b, c, d\}$. In Step 1 of Π-**LangInt**, we initialize Π to the empty set. Then in Step 2, we examine each of the conjuncts of φ one by one:

- We first examine $a \vee \Box_2 \Box_1 d$. As this clause has a propositional disjunct a and $a \notin \mathcal{L}$, we do not enter the if-loop.
- The next conjunct is $\neg a \vee \Diamond_2 (\, a \wedge b \,)$. Again, the conditions of the if-loop are not satisfied, as there is a disjunct $\neg a$ and $a \notin \mathcal{L}$.
- We next consider the conjunct $\Box_2 (\, (\, \Box_1 d \vee a \,) \wedge (\, \Box_1 d \vee b \,) \,) \vee \Diamond_2 (\, a \wedge b \,))$. As there are no propositional disjuncts and $2 \in \mathcal{L}$, the conditions of the if-loop are satisfied. We thus make recursive calls to Π-**LangInt** on the formulae behind the modal operators:
  - On input $(\, \Box_1 d \vee a \,) \wedge (\, \Box_1 d \vee b \,)$, Π-**LangInt** returns $\Box_1 d \vee b$, since the first conjunct does not satisfy the conditions of the if-loop because of its disjunct a, and the second conjunct is not modified as Π-**LangInt**$(d,\mathcal{L})$=d.
  - On input $a \wedge b$, Π-**LangInt** returns b, since a does not satisfy the conditions of the if-loop, and b is left unaltered.

  We thereby replace $(\, \Box_1 d \vee a \,) \wedge (\, \Box_1 d \vee b \,)$ by $\Box_1 d \vee b$ and $a \wedge b$ by b. The clause resulting from these modifications is then added to Π.

- The next conjunct of φ is $\Box_1((a \vee \Diamond_2 \top) \wedge (\Box_2 \bot \vee c) \wedge (a \vee c))$. As this clause has no propositional disjuncts and $1 \in \mathcal{L}$, we enter the if-loop and call Π-**LangInt** on the formula behind the modal operator.
  - On input $(a \vee \Diamond_2 \top) \wedge (\Box_2 \bot \vee c) \wedge (a \vee c)$, Π-**LangInt** returns $\Box_2 \bot \vee c$ since the first and third clauses do not satisfy the conditions of the if-loop because of their disjunct a, and the second clause is not modified as Π-**LangInt**$(\bot,\mathcal{L})=\bot$.

  We thus add $\Box_1(\Box_2 \bot \vee c)$ to Π.
- The final conjunct is $\Diamond_1(a \wedge \neg c \wedge \Box_2 \bot)$. As $1 \in \mathcal{L}$, we enter the if-loop, and we call Π-**LangInt** on $a \wedge \neg c \Box_2 \bot$:
  - On input $a \wedge \neg c \wedge \Box_2 \bot$, Π-**LangInt** returns $\neg c \wedge \Box_2 \bot$, since the first conjunct does not satisfy the conditions of the if-loop, and the other two conjuncts are left unaltered.

  We thus add $\Diamond_1(\neg c \wedge \Box_2 \bot)$ to Π.

In Step 3, we return the conjunction of the elements of Π, which is:

$$(\Box_2(\Box_1 d \vee b) \vee \Diamond_2 b) \wedge \Box_1(\Box_2 \bot \vee c) \wedge \Diamond_1(\neg c \wedge \Box_2 \bot)$$

We now prove the correctness of Π-**LangInt**.

Lemma 6.3.20.
If φ is a formula in prime implicate normal form, then the output of Π-**LangInt**$(\varphi, \mathcal{L})$ *is an $\mathcal{L}$-interpolant of φ.*

Proof. The proof is by induction on the depth of the input formula φ. The base case is when $\delta(\varphi) = 0$, i.e. when φ is a propositional formula. In this case, the algorithm simply returns the conjunction of the conjuncts of φ whose signatures are contained in $\mathcal{L}$, or $\top$ if there are no such conjuncts. Let ζ be some formula such that $sig(\zeta) \subseteq \mathcal{L}$ and $\varphi \models \zeta$. Because of Theorem 3.1.13, we can suppose without loss of generality that ζ is a conjunction of clauses. We know from **Covering** (Theorem 3.2.8) that if a clause λ is entailed by φ, then there is some prime implicate of φ, hence some conjunct of φ (since φ is in prime implicate normal form), which entails λ. It follows that every conjunct of ζ is entailed by some conjunct of φ. We remark that φ is a conjunction of propositional clauses and that a propositional clause containing propositional literals outside $\mathcal{L}$ cannot entail a non-tautologous formula with signature contained in $\mathcal{L}$. This means that if there are no conjuncts of φ with signature contained in $\mathcal{L}$, then ζ must be a tautological formula, and if such conjuncts exist, then each of the conjuncts in ζ must be entailed by at least one such conjunct. In the first case, we find that ζ is entailed by $\top$, and in the second case, ζ is entailed by the conjunction of the conjuncts of φ whose signatures

are contained in $\mathcal{L}$. In both cases, we find that the output of Π-**LangInt**$(\varphi, \mathcal{L})$ is a formula with signature in $\mathcal{L}$ which is entailed by φ and which entails every formula ζ in $\mathcal{L}$ which is entailed by φ, so it must be an $\mathcal{L}$-interpolant of φ.

Let us next assume that the result holds for every formula in prime implicate normal form with depth at most k and show that the result still holds for formulae with depth $k+1$. Our first step will be to show that the following statements hold for every clause λ in prime implicate normal form with depth at most $k+1$:

1. If λ contains a disjunct $(\neg)a$ with $a \notin \mathcal{L}$ or a disjunct $\Box_i\psi$ or $\Diamond_i\psi$ where $i \notin \mathcal{L}$, then $\top$ is an $\mathcal{L}$-interpolant of λ
2. If λ is such that $a \in \mathcal{L}$ for every disjunct $(\neg)a$ of λ and $i \in \mathcal{L}$ for every disjunct $\Box_i\psi$ or $\Diamond_i\psi$, then the formula obtained from λ by replacing disjuncts of the form $\Diamond_i\psi$ and $\Box_i\psi$ respectively by $\Diamond_i \Pi$-**LangInt**$(\psi, \mathcal{L})$ and $\Box_i \Pi$-**LangInt**$(\psi, \mathcal{L})$ is an $\mathcal{L}$-interpolant of λ

We begin with statement (1). Let λ be a clause in prime implicate normal form which contains a disjunct $(\neg)a$ with $a \notin \mathcal{L}$ or a disjunct $\Box_i\psi$ or $\Diamond_i\psi$ where $i \notin \mathcal{L}$. In the first case, we know that the only formulae with signature in $\mathcal{L}$ which subsume $(\neg)a$ are tautologous formulae, so λ cannot entail any non-tautologous formulae with signature contained in $\mathcal{L}$, i.e. $\top$ is an $\mathcal{L}$-interpolant of λ. If instead we are in the second case, then there is some satisfiable disjunct $\Diamond_i\psi$ or $\Box_i\psi$ of λ with $i \notin \mathcal{L}$. But a satisfiable formula $\Diamond_i\psi$ or $\Box_i\psi$ cannot imply any non-tautological clause which does not contain a modality $\Diamond_i$ or $\Box_i$ by Theorem 2.3.3. It follows that every formula which is entailed by λ and has signature contained in $\mathcal{L}$ is tautologous, so $\top$ is an $\mathcal{L}$-interpolant of λ.

We now show (2). Let λ be a clause in prime implicate normal form of depth at most $k+1$ such that $a \in \mathcal{L}$ for every disjunct $(\neg)a$ of λ and $i \in \mathcal{L}$ for every disjunct $\Box_i\psi$ or $\Diamond_i\psi$ of λ. Let λ' be the formula obtained from λ by replacing disjuncts of the form $\Diamond_i\psi$ and $\Box_i\psi$ respectively by $\Diamond_i\Pi$-**LangInt**$(\psi, \mathcal{L})$ and $\Box_i\Pi$-**LangInt**$(\psi, \mathcal{L})$. We remark that λ' has the same propositional disjuncts as λ, and its top-level modalities are the same as those in λ. We also note that the formulae appearing behind the top-level modal operators have the form Π-**LangInt**$(\psi, \mathcal{L})$ where ψ is a formula in prime implicate normal form with depth at most k. Applying the induction hypothesis, we find that for each such formula ψ, Π-**LangInt**$(\psi, \mathcal{L})$ is an $\mathcal{L}$-interpolant of ψ. In particular, that means that Π-**LangInt**$(\psi, \mathcal{L})$ has signature contained in $\mathcal{L}$. It follows that ψ' also has signature contained in $\mathcal{L}$. It also means that each ψ entails Π-**LangInt**$(\psi, \mathcal{L})$, from which we can deduce that λ' is entailed by λ. We now need to show that λ' entails every formula which is entailed by λ and has signature in $\mathcal{L}$. Let γ be such a formula. Clearly every propositional disjunct of λ' must entail γ since λ and λ' have the same propositional disjuncts and

$\lambda \models \gamma$. Every existential disjunct of λ' is equal to $\Diamond_i \Pi$-**LangInt**$(\psi, \mathcal{L})$ for some disjunct $\Diamond_i \psi$ of λ. As $\Diamond_i \Pi$-**LangInt**$(\psi, \mathcal{L})$ is an $\mathcal{L}$-interpolant of $\Diamond_i \psi$ (Lemma 2.6.10), it follows that $\Diamond_i \Pi$-**LangInt**$(\psi, \mathcal{L})$ must entail γ since γ is entailed by $\Diamond_i \psi$ and $sig(\gamma) \subseteq \mathcal{L}$. That means that every $\Diamond$-disjunct of λ' entails γ. Likewise, we remark that $\Box_i \Pi$-**LangInt**$(\psi, \mathcal{L})$ is an $\mathcal{L}$-interpolant of $\Box_i \psi$, so every $\Box$-disjunct of λ' entails γ. As every disjunct of λ' entails γ, it follows that $\lambda' \models \gamma$, so λ' is an $\mathcal{L}$-interpolant of λ. This together with statement (1) tells us that the output of Π-**LangInt**$(\varphi,\ k+1)$ is equivalent to the conjunction of $\mathcal{L}$-interpolants of the conjuncts of φ.

Now let ζ be a formula such that $sig(\zeta) \subseteq \mathcal{L}$ and $\varphi \models \zeta$. We can assume without loss of generality that ζ is a conjunction of clauses since any formula is equivalent to some formula of this form and with the same or smaller signature (Theorem 2.4.11). Then since φ entails ζ, it follows that φ entails each of the clauses which are conjuncts of ζ. By the **Covering** property (Theorem 3.2.8), we know that every implicate of φ is entailed by some prime implicate of φ. As we have assumed φ to be in prime implicate normal form, this means that every conjunct of ζ is entailed by some conjunct of φ. This together with the fact that the conjuncts of ζ have signature contained in $\mathcal{L}$ means that each of the conjuncts of ζ is entailed by an $\mathcal{L}$-interpolant of some conjunct of φ. It follows then that the formula ζ is entailed by the conjunction of the $\mathcal{L}$-interpolants of the conjuncts of φ. But then ζ must be entailed by the output of Π-**LangInt**$(\varphi,\ \mathcal{L})$ since we have shown above that the output of Π-**LangInt**$(\varphi,\ \mathcal{L})$ is equivalent to the conjunction of $\mathcal{L}$-interpolants of the conjuncts of φ. We have thus demonstrated that the output of Π-**LangInt**$(\varphi,\ \mathcal{L})$ is a formula with signature in $\mathcal{L}$ which is entailed by φ and entails every formula with signature in $\mathcal{L}$ which is entailed by φ, i.e. the output of Π-**LangInt**$(\varphi,\ \mathcal{L})$ is an $\mathcal{L}$-interpolant of φ. □

Lemma 6.3.21.
The algorithm Π-**LangInt** *runs in linear time in the size of the input formula.*

Proof. Π-**LangInt** terminates in linear time with respect to the size of the input formula because all the algorithm does is scan the input formula a single time in order to remove those clausal sub-formulae which violate the syntactic requirements set forth in Step 2. □

Theorem 6.3.22.
If φ is in prime implicate normal form, then an $\mathcal{L}$-interpolant of φ can be generated in linear time in the size of φ.

Proof. Follows directly from Lemmas 6.3.20 and 6.3.21. □

The $\mathcal{L}$-interpolant obtained using Π-**LangInt** may not, however, be itself in prime implicate normal form, as the following example demonstrates:

Example 6.3.23.
Let $\varphi = \Diamond_1(a \wedge \Box_2 b) \wedge \Diamond_1(a \wedge b)$ and $\mathcal{L} = \{1, a, b\}$. Then φ is in prime implicate normal form, but Π-**LangInt**$(\varphi, \mathcal{L}) = \Diamond_1 a \wedge \Diamond_1(a \wedge b)$ is not in prime implicate normal form since $\Diamond_1 a$ is not a prime implicate of Π-**LangInt**$(\varphi, \mathcal{L})$.

This is not a problem, however, since we show in Theorem 6.3.25 that we can use the algorithm Π-**Entail** from earlier in the chapter in order to remove superfluous subformulae, thereby returning the $\mathcal{L}$-interpolant to prime implicate normal form. For the proof of Theorem 6.3.25, we will require the following lemma, which shows that the output of Π-**LangInt** satisfies the required conditions to ensure the correctness of Π-**Entail**. Both the proof of the lemma and the proof of Theorem 6.3.25 are quite long and tedious, and so might be best skipped on a first reading of the chapter.

Lemma 6.3.24.
If φ is a formula in prime implicate normal form, then the output of Π-**LangInt**$(\varphi, \mathcal{L})$ *satisfies the requirements for both left- and right-hand-side formulae described in Theorem 6.3.9.*

Proof. The proof is by induction on the depth of the input formula φ. The base case is when $\delta(\varphi) = 0$. In this case, Π-**LangInt** simply returns the conjunction of the conjuncts of φ whose signatures are contained in $\mathcal{L}$, or $\top$ if there are no such conjuncts. We consider only the former case, since $\top$ clearly satisfies all of the conditions for both right- and left-hand-side formulae. Now let λ be some prime implicate of Π-**LangInt**$(\varphi, \mathcal{L})$. We know that λ is non-tautologous since Π-**LangInt**$(\varphi, \mathcal{L})$ contains at least one non-tautologous propositional clause. Moreover, since the signature of Π-**LangInt**$(\varphi, \mathcal{L})$ is contained in $\mathcal{L}$, then by Theorem 4.1.5, we can assume without loss of generality that λ also has signature contained in $\mathcal{L}$. As $\varphi \models \Pi$-**LangInt**$(\varphi, \mathcal{L})$, we must also have $\varphi \models \lambda$, so by the **Covering** property (Theorem 3.2.8), there must be some prime implicate of φ which entails λ. As φ is in prime implicate normal form, every prime implicate is equivalent to some conjunct of φ, so there must be some conjunct π of φ such that $\pi \models \lambda$. But then by Theorem 2.3.3, it follows that π must have signature in $\mathcal{L}$, which means that π is a conjunct of Π-**LangInt**$(\varphi, \mathcal{L})$, and also that $\pi \equiv \lambda$. It follows that every prime implicate of Π-**LangInt**$(\varphi, \mathcal{L})$ is equivalent to some conjunct of Π-**LangInt**$(\varphi, \mathcal{L})$. As Π-**LangInt**$(\varphi, \mathcal{L})$ had depth 0, the second condition for left-hand-side formulae is trivially satisfied. As Π-**LangInt**$(\varphi, \mathcal{L})$ is a propositional formula, there are

only two possible types of unsatisfiable subformulae: Π-**LangInt**$(\varphi, \mathcal{L})$ itself, or disjuncts of the form $\bot$. In the first case, we have Π-**LangInt**$(\varphi, \mathcal{L}) = \bot$, since Π-**LangInt**$(\varphi, \mathcal{L})$ is unsatisfiable only if φ is, φ would be equal to $\bot$ if it were unsatisfiable, and Π-**LangInt**$(\bot) = \bot$. In the second case, we would have had disjuncts $\bot$ in φ, which cannot be the case since φ is assumed to be in prime implicate normal form. Finally, we note that Π-**LangInt**$(\varphi, \mathcal{L})$ is a conjunction of clauses, and hence in extended conjunctive normal form. We have thus shown that Π-**LangInt**$(\varphi, \mathcal{L})$ satisfies all of the requirements for left-hand-side formulae.

Now let's show that Π-**LangInt**$(\varphi, \mathcal{L})$ satisfies the requirements for right-hand-side formulae. As Π-**LangInt**$(\varphi, \mathcal{L})$ had depth 0, the first two requirements, which involve $\Box$- and $\Diamond$-formulae, are trivially satisfied. The third condition, that all tautologous subformulae are equal to $\top$, is also satisfied, since Π-**LangInt**$(\varphi, \mathcal{L})$ is either equal to $\top$ or it is a conjunction of non-tautologous propositional clauses. The final requirement is satisfied as well, since Π-**LangInt**$(\varphi, \mathcal{L})$ cannot have any tautologous conjuncts unless it is itself a tautology.

We have thus shown that Π-**LangInt**$(\varphi, \mathcal{L})$ satisfies the requirements for both left- and right-hand-side formulae whenever φ has depth 0. Now let us suppose that the same holds for formulae with depth at most m, and let φ be a formula in prime implicate normal form of depth $m+1$. We begin with the left-hand-side requirements. Let λ be some prime implicate of Π-**LangInt**$(\varphi, \mathcal{L})$. We can assume without loss of generality that $sig(\lambda) \subseteq \mathcal{L}$ since the signature of Π-**LangInt**$(\varphi, \mathcal{L})$ is contained in $\mathcal{L}$, and we know that every prime implicate of Π-**LangInt**$(\varphi, \mathcal{L})$ is equivalent to some clause with signature in sig(Π-**LangInt**$(\varphi, \mathcal{L})$) (by Theorem 4.1.5). Now since Π-**LangInt**$(\varphi, \mathcal{L}) \models \lambda$ and $\varphi \models \Pi$-**LangInt**$(\varphi, \mathcal{L})$, we also have $\varphi \models \lambda$. By the **Covering** property (Theorem 3.2.8), there must be some prime implicate of φ, and hence some conjunct of φ (since φ is in prime implicate normal form), which entails λ. We showed in the proof of Theorem 6.3.20 that Π-**LangInt** transforms every conjunct of φ into its $\mathcal{L}$-interpolant, from which we find that λ is equivalent to some conjunct of Π-**LangInt**$(\varphi, \mathcal{L})$.

Next let ψ be such that $\Diamond_k \psi$ or $\Box_k \psi$ is a subformula of Π-**LangInt**$(\varphi, \mathcal{L})$. Then it must be the case that $\psi = \Pi$-**LangInt**$(\zeta, \mathcal{L})$ for some subformula ζ of φ such that $\Diamond_k \zeta$ or $\Box_k \zeta$ is a subformula of φ. But that means that ζ must be a formula in prime implicate normal form with depth at most m, so applying the induction hypothesis, we find that Π-**LangInt**$(\zeta, \mathcal{L})$ satisfies all of the left-hand-side requirements. In particular, this means that if ψ is such that $\Diamond_k \psi$ or $\Box_k \psi$ is a subformula of Π-**LangInt**$(\varphi, \mathcal{L})$, then every prime implicate of ψ is equivalent to some conjunct of ψ. It also means that ψ is in extended conjunctive normal form, which allows us to show that Π-**LangInt**$(\varphi, \mathcal{L})$ is also in extended conjunctive normal form.

Let us now show that the third left-hand-side condition holds for Π-**LangInt**(φ, $\mathcal{L}$). Suppose then for a contradiction that Π-**LangInt**(φ, $\mathcal{L}$) has an unsatisfiable subformula γ such that $\gamma \neq \bot$. Suppose first that γ appears in the scope of one or more modal operators. Then that means that there is some disjunct $\Diamond_k\psi$ or $\Box_k\psi$ of one of the clausal conjuncts of Π-**LangInt**(φ, $\mathcal{L}$), such that γ is a subformula of ψ. But we know from the definition of Π-**LangInt** that $\psi = \Pi$-**LangInt**(ζ, $\mathcal{L}$) for some ζ such that $\Diamond_k\psi$ or $\Box_k\psi$ is a subformula of φ. By applying the induction hypothesis to ζ, we conclude that $\psi = \Pi$-**LangInt**(ζ, $\mathcal{L}$) satisfies all of the left-hand-side conditions. In particular, it contains no unsatisfiable subformulae which are not equal to $\bot$. It follows that γ must appear in Π-**LangInt**(φ, $\mathcal{L}$) outside the scope of the modal operators. There are three possibilities: γ is a disjunct of some clausal conjunct of Π-**LangInt**(φ, $\mathcal{L}$), γ is a clausal conjunct of Π-**LangInt**(φ, $\mathcal{L}$), or γ is a conjunction of clausal conjuncts of Π-**LangInt**(φ, $\mathcal{L}$). We start with the case where γ is a disjunct of some clausal conjunct of Π-**LangInt**(φ, $\mathcal{L}$). In this case, γ must be of the form $\Diamond_k\psi$, since propositional literals and $\Box$-formulae cannot be unsatisfiable. We know from the definition of Π-**LangInt** that $\psi = \Pi$-**LangInt**(ζ, $\mathcal{L}$) for some subformula ζ of φ such that $\Diamond_k\zeta$ is a disjunct of some conjunct of φ. But ψ is both unsatisfiable and an $\mathcal{L}$-interpolant of ζ (by Lemma 6.3.20), so ζ must be unsatisfiable, too. That means that φ possesses a clausal conjunct with an unsatisfiable disjunct, contradicting our assumption that φ is in prime implicate normal form. Next consider the possibility that γ is a clausal conjunct of Π-**LangInt**(φ, $\mathcal{L}$). Since γ is assumed unsatisfiable but not equal to $\bot$, it must either contain a disjunct of the form $\Diamond_k\psi$ for some unsatisfiable formula ψ, or multiple disjuncts of the form $\bot$. We have already shown that the first case cannot occur, and the second case is also impossible, since it would mean that φ possesses a clausal conjunct with unnecessary disjuncts. It must then be the case that Π-**LangInt**(φ, $\mathcal{L}$) itself is unsatisfiable but unequal to $\bot$. But that would imply the presence of either an unsatisfiable conjunct not equal to $\bot$, which we have just shown to be impossible, or several conjuncts $\bot$, which is also impossible since that would mean that φ itself contains unnecessary conjuncts, which cannot happen since φ is in prime implicate normal form.

Finally, for the fourth left-hand-side condition, we simply note that by the discussion in the previous paragraph, there can be no satisfiable clausal subformula in Π-**LangInt**(φ, $\mathcal{L}$) having unsatisfiable disjuncts, as this would indicate the presence of unnecessary disjuncts in some clausal subformula of φ, which is forbidden by the definition of prime implicate normal form.

Now let us move on to the right-hand-side requirements. We know from the induction hypothesis that any clausal subformula appearing within the scope of

modal operators in Π-**LangInt**(φ, $\mathcal{L}$) must satisfy the first two right-hand-side conditions. So we only need to worry about the clausal subformulae which are conjunct of Π-**LangInt**(φ, $\mathcal{L}$). Let λ' be a clausal conjunct of Π-**LangInt**(φ, $\mathcal{L}$). We know from the definition of Π-**LangInt** that there must be some clausal conjunct λ of φ such that λ' is obtained from λ by replacing each disjunct $\Box_k\psi$ by $\Box_k$Π-**LangInt**($\psi, \mathcal{L}$) and each disjunct $\Diamond_k\psi$ by $\Diamond_k$Π-**LangInt**($\psi, \mathcal{L}$). As φ is in prime implicate normal form, it follows that $|Diam_k(\lambda)| \leq 1$ for all $1 \leq k \leq n$. It follows then that $|Diam_k(\lambda')| \leq 1$ for all $1 \leq k \leq n$, so the first right-hand-side condition is satisfied by Π-**LangInt**(φ, $\mathcal{L}$). Now suppose that $\gamma' \in Diam_k(\lambda')$ and $\zeta' \in Box_k(\lambda')$ for some k. Then there must be $\gamma \in Diam_k(\lambda)$ and $\zeta \in Box_k(\lambda)$ such that $\gamma' =$Π-**LangInt**(γ, $\mathcal{L}$) and $\zeta' =$Π-**LangInt**(ζ, $\mathcal{L}$). As φ is assumed to be in prime implicate normal form, we know that $\gamma \models \zeta$. But that means that Π-**LangInt**(γ, $\mathcal{L}$)$\models$Π-**LangInt**(ζ, $\mathcal{L}$), and hence $\gamma' \models \zeta'$. We thus have the second right-hand-side condition.

Next we need to show that every tautologous subformula of Π-**LangInt**(φ, $\mathcal{L}$) is equal to $\top$. Suppose for a contradiction that Π-**LangInt**(φ, $\mathcal{L}$) contains some tautologous subformula γ which is not equal to $\top$. It follows from the induction hypothesis that all tautologous formulae appearing behind the modal operators are equal to $\top$, so γ must appear outside the scope of any modal operators in Π-**LangInt**(φ, $\mathcal{L}$). There are three possibilities: γ is a disjunct of some clausal conjunct of Π-**LangInt**(φ, $\mathcal{L}$), γ is a clausal conjunct of Π-**LangInt**(φ, $\mathcal{L}$), or γ is a conjunction of clausal conjuncts of Π-**LangInt**(φ, $\mathcal{L}$). First consider the case where γ is a literal. Then γ must be of the form $\Box_k\zeta'$ where ζ' is a tautologous formula, since propositional literals and $\Diamond$-formulae cannot be tautologies. But we know from the definition of Π-**LangInt** that $\Box_k\zeta'$ must be equal to $\Box_k$Π-**LangInt**(ζ, $\mathcal{L}$) for some ζ such that $\Box_k\zeta$ is a disjunct of some clausal conjunct of φ. By the induction hypothesis, Π-**LangInt**(ζ, $\mathcal{L}$) must be equal to $\top$ since it is a tautology, which means that γ is of the form $\Box_k\top$. But this is a contradiction, since in Step 2(b) of Π-**LangInt** we only added to the set Π clauses which did not contain disjuncts of this form. Next suppose that γ is one of the clausal conjuncts of Π-**LangInt**(φ, $\mathcal{L}$). Since γ is a tautologous clause, and we know from the previous paragraph that $|Diam_k(\lambda')| \leq 1$, it follows from Theorem 2.3.1 that γ contains either a pair of complementary literals a and $\neg a$, or a pair disjuncts $\Box_k\zeta'$ and $\Diamond_k\mu'$ such that $\models \zeta' \vee \mu'$. We know that the former case cannot occur, since in Step 2 of Π-**LangInt** we do not add to Π any clauses with complementary propositional disjuncts. It must then be the case that γ possesses disjuncts $\Box_k\zeta'$ and $\Diamond_k\mu'$ such that $\models \zeta' \vee \mu'$. From the previous paragraph, we know that $\mu' \models \zeta'$, which gives us $\models \zeta'$. But we also know that $\Box_k\zeta'$ must be equal to $\Box_k$Π-**LangInt**(ζ, $\mathcal{L}$) for

some ζ such that $\Box_k\zeta$ is a disjunct of some clausal conjunct of φ. But we know from the induction hypothesis that Π-**LangInt**$(\zeta, \mathcal{L})$ cannot have any tautologous subformulae not equal to $\top$, so we must have Π-**LangInt**$(\zeta, \mathcal{L}) = \top$ and hence $\Box_k\zeta' = \Box_k\top$. This is a contradiction however, since no clauses with a disjunct of the form $\Box_k\top$ are added to Π in Step 2. Thus, we must be in the final case, in which γ is a conjunction of clausal conjuncts of Π-**LangInt**$(\varphi, \mathcal{L})$. But then γ must either have a tautologous clausal conjunct which is not equal to $\top$, or be a conjunction of multiple $\top$ symbols. We just showed that Π-**LangInt**$(\varphi, \mathcal{L})$ cannot have a tautologous clausal conjunct not equal to $\top$. The second case also cannot happen, since that would mean that we had added $\top$ to Π in Step 2. But that could only happen if $\top$ were a conjunct of φ, which can only happen when $\varphi = \top$, in which case we would have Π-**LangInt**$(\varphi, \mathcal{L}) = \top$. Thus, we have shown that Π-**LangInt**$(\varphi, \mathcal{L})$ cannot possess any tautologous subformulae not equal to $\top$.

For the final right-hand-side condition, suppose that Π-**LangInt**$(\varphi, \mathcal{L})$ possesses a non-tautologous subformula which is a conjunction having at least one conjunct $\top$. Because of the induction hypothesis, we can assume that this conjunction does not appear behind the scope of modal operators, which means that the conjunct in question must be a conjunct of Π-**LangInt**$(\varphi, \mathcal{L})$. But we showed at the end of the last paragraph that Π-**LangInt**$(\varphi, \mathcal{L})$ cannot have a conjunct $\top$ except in the limit case where it is itself equal to $\top$. Thus, we have shown Π-**LangInt**$(\varphi, \mathcal{L})$ to satisfy all of the right-hand-side conditions. □

Theorem 6.3.25.
If φ is in prime implicate normal form, then an $\mathcal{L}$-interpolant of φ in prime implicate normal form can be generated in polynomial time in the size of φ.

Proof. We know from Lemma 6.3.20 that we can compute an $\mathcal{L}$-interpolant of a formula φ in prime implicate normal form using the algorithm Π-**LangInt**, but we now need to show how to transform the output of Π-**LangInt** into an equivalent formula in prime implicate normal form. Consider the following recursive procedure **BackToPINF** which takes as input a formula $\zeta = \lambda_1 \wedge ... \wedge \lambda_m$ in extended conjunctive normal form:

Step 1 If $\delta(\zeta) = 0$, return ζ. Otherwise, set $\Pi = \emptyset$.

Step 2 For $i = 1$ to m:

If Π-**Entail**(λ_j, λ_i)=**no** for all $1 \leq j < i$ *and* either Π-**Entail**(λ_j, λ_i)=**no** or Π-**Entail**(λ_i, λ_j)=**yes** for all $i < j \leq n$, then

(a) Let λ_i' be the formula obtained from λ_i by replacing each

disjunct $\Box_k\psi$ (resp. $\Diamond_k\psi$) with $\Box_k$**BackToPINF**(ψ) (resp. $\Diamond_k$**BackToPINF**(ψ)).

(b) For each $k = 1$ to n:

Let $\{\alpha_1, ..., \alpha_s\}$ be the elements in $Box_k(\lambda_i')$.

For $l = 1$ to s:

If Π-**Entail**(α_l, α_p)=**yes** for some $1 \leq p < l$ *or* Π-**Entail**(α_l, α_p)=**yes** and Π-**Entail**(α_p, α_l)=**no** for some $l < p \leq s$, then

Remove $\Box_k\alpha_l$ from λ_i'.

(c) Add λ_i' to Π.

Step 3 Return $\bigwedge_{\pi\in\Pi}\pi$.

We claim that if ζ is such that $\zeta = \Pi$-**LangInt**$(\varphi, \mathcal{L})$ for some formula φ in prime implicate normal form and some signature $\mathcal{L}$, then the output of **BackToPINF** on input ζ, call it ζ', is a formula in prime implicate normal form which is equivalent to ζ. The proof is by induction on the depth of the input formula ζ.

The base case is when $\delta(\zeta) = 0$. Let φ be a formula in prime implicate normal form such that $\zeta = \Pi$-**LangInt**$(\varphi, \mathcal{L})$. Now since ζ is propositional, we know that φ must be propositional as well, since Π-**LangInt** never removes modalities from clauses. For propositional formulae, it is known that the $\mathcal{L}$-prime implicates of a formula are precisely those prime implicates of the formula with signature contained in $\mathcal{L}$ (cf. [Mar00]). It follows that the conjuncts of ζ are precisely its prime implicates. Moreover, there can be no repeated conjuncts or disjuncts in ζ since this would imply unnecessary conjuncts or disjuncts in φ, which cannot happen since φ is in prime implicate normal form. It follows that ζ is itself in prime implicate normal form. We thus have the desired result since **BackToPINF**(ζ)=ζ for propositional ζ.

Next let us assume that the result holds whenever the input formula has the required form and depth at most d, and let ζ a formula of depth $d + 1$ such that $\zeta = \Pi$-**LangInt**$(\varphi, \mathcal{L})$ for some formula φ in prime implicate normal form and some signature $\mathcal{L}$. We begin by showing that every conjunct in ζ is implied by some conjunct of ζ'. Let λ_i be a conjunct of ζ. We begin with the case where λ_i satisfies the conditions of the if-loop. Then in Step 2(a), we will set λ_i' equal to the clause obtained from λ_i by replacing each disjunct $\Box_k\psi$ (resp. $\Diamond_k\psi$) with $\Box_k$**BackToPINF**(ψ) (resp. $\Diamond_k$**BackToPINF**(ψ)). We know that the disjunct $\Box_k\psi$ (resp. $\Diamond_k\psi$) must be equal to $\Box_k\Pi$-**LangInt**$(\gamma,\mathcal{L})$ (resp. $\Diamond_k\Pi$-**LangInt**$(\gamma,\mathcal{L})$) where $\Box_k\gamma$ (resp. $\Diamond_k\gamma$) is a subformula of φ. It follows then by the induction

hypothesis that **BackToPINF**(ψ) is equivalent to ψ, which means that $\lambda'_i \equiv \lambda_i$ at the end of Step 2(a). We just need to ensure that the changes in Step 2(b) are equivalence-preserving. To do so, we show that every disjunct of λ'_i which is removed during Step 2(b) implies one of the disjuncts of λ'_i which remains at the end of Step 2(b). We only need to show this for $\Box$-disjuncts, since we do not remove any other disjuncts in Step 2(b). We note that by the induction hypothesis the formulae in $Box_k(\lambda'_i)$ must all be in prime implicate normal form. It follows then by Lemmas 6.3.2 and 6.3.3 that for $\alpha_q, \alpha_r \in Box_k(\lambda'_i)$, we have Π-**Entail**(α_q, α_r) = **yes** if and only if $\alpha_q \models \alpha_r$. Now let $\Box_k\alpha_l$ be a disjunct of λ'_i which was removed in Step 2(b). That means that α_l satisfies the conditions of the if-loop, so there must be either some $p < l$ such that Π-**Entail**(α_l, α_p)=**yes** or some $p > l$ such that both Π-**Entail**(α_l, α_p)=**yes** and Π-**Entail**(α_p, α_l)=**no**. It follows then that either $\alpha_l \models \alpha_p$ for some $p < l$, or both $\alpha_l \models \alpha_p$ and $\alpha_p \not\models \alpha_l$ for some $p > l$. This means there is some p such that $\alpha_l \models \alpha_p$ and $\alpha_p \not\models \alpha_q$ for $q < p$ and $\alpha_p \models \alpha_q$ only if $\alpha_q \models \alpha_p$ for $q > p$. But then we have Π-**Entail**(α_p, α_q)=**no** for every $q < p$, and either Π-**Entail**(α_p, α_q)=**no** or Π-**Entail**(α_q, α_p)=**yes** for $q > p$. This means that we will not enter the if-loop on α_p, so α_p will not be removed from λ'_i in Step 2(b). We have thus found a disjunct of λ'_i at the end of Step 2(b) which is entailed by α_l. It follows that the modifications in 2(b) are equivalence-preserving. So the conjunct λ'_i of ζ' is equivalent to the conjunct λ_i of ζ.

Next let us consider the other case in which we do not enter the if-loop when examining the conjunct λ_i of ζ. Then that means that either there is some $j < i$ such that Π-**Entail**(λ_j, λ_i)=**yes**, or there is some $j > i$ such that Π-**Entail**(λ_j, λ_i)=**yes** and Π-**Entail**(λ_i, λ_j)=**no**. Now we know that ζ satisfies the conditions on both right- and left-hand-side formulae by Lemma 6.3.24. It follows that the conjuncts of ζ also satisfy these conditions, so by Theorem 6.3.9, Π-**Entail** will give the right answer on entailment queries concerning conjuncts of ζ. We thus find that either $\lambda_j \models \lambda_i$ for some $j < i$, or $\lambda_j \models \lambda_i \not\models \lambda_j$ for some $j > i$. We can thus choose some index j such that (i) $\lambda_j \models \lambda_i$, (ii) $\lambda_l \not\models \lambda_j$ for all $l < j$, and (iii) for every $l > j$ either $\lambda_l \not\models \lambda_j$ or $\lambda_j \models \lambda_l$. But then λ_j will satisfy all of the requirements for entering the if-loop, so we know from above that there must be some conjunct of ζ' which is equivalent to λ_j, and hence some conjunct of ζ' which entails λ_i. Thus, every conjunct of ζ is implied by some conjunct of ζ'. It follows that $\zeta' \models \zeta$. As we have also showed that every conjunct of ζ' is equivalent to some conjunct of ζ, we must also have $\zeta \models \zeta'$, hence $\zeta \equiv \zeta'$.

We will now prove that ζ' is in prime implicate normal form. We first remark that if ζ' is unsatisfiable, then so is ζ. Since ζ satisfies all left-hand-side conditions, it can have not unsatisfiable subformulae not equal to $\bot$, which means $\zeta = \bot$, and

hence $\zeta' = \bot$, as desired. Likewise, since ζ satisfies all right-hand-side conditions, if ζ is tautologous, then it must be equal to $\top$. We now consider the case where ζ is neither tautologous nor unsatisfiable.

We first need to show that all prime implicates of ζ' are equivalent to some conjunct of ζ'. As ζ satisfies right- and left-hand-side conditions (by Lemma 6.3.24), we know that ζ is a conjunction of clauses. As the modifications to conjuncts of ζ in Step 2 of **BackToPINF** leave clauses as clauses, ζ' must also be a conjunction of clauses. Let us then consider some prime implicate π of ζ'. We know that $\zeta \equiv \zeta'$ from above, so it must be the case that π is also a prime implicate of ζ, and hence equivalent to some conjunct of ζ (because ζ satisfies all left-hand-side conditions). Let λ_i be a conjunct of ζ such that $\pi \equiv \lambda_i$ and $\pi \not\equiv \lambda_j$ for $j < i$. Now we have seen earlier that all of the conjuncts of ζ must satisfy all of the right- and left-hand-side conditions. It follows then from Theorem 6.3.9 that the algorithm **Π-Entail** gives the correct output when run on pairs of conjuncts of ζ. This means in particular that **Π-Entail**$(\lambda_j,\ \lambda_i)$=**no** for all $1 \leq j < i$ and either **Π-Entail**$(\lambda_j,\ \lambda_i)$=**no** or **Π-Entail**$(\lambda_i,\ \lambda_j)$=**yes** for every $i < j \leq n$. But then we must enter the main if-loop when we examine λ_i in Step 2. So we will add a clause λ'_i to Π, making λ'_i a conjunct of ζ'. We have seen above that the clause λ'_i must be equivalent to λ_i, so we have $\pi \equiv \lambda'_i$. We have thus shown that every prime implicate of ζ' is equivalent to one of the conjuncts of ζ'.

Let us now show that ζ' does not contain any unnecessary conjuncts. Suppose for a contradiction that this is the case, i.e. there are conjuncts λ'_i and λ'_j with $\lambda'_i \models \lambda'_j$ and $i \neq j$. Now we know that each conjunct λ'_l in ζ' was obtained from a corresponding conjunct λ_l in ζ via the modifications in Step 2. These modifications are equivalence-preserving (see above), so we must also have $\lambda'_l \equiv \lambda_l$ for every conjunct λ'_l of ζ'. This means in particular that $\lambda_i \models \lambda_j$. We consider two cases: $\lambda_j \models \lambda_i$ or $\lambda_j \not\models \lambda_i$. We begin with the case where $\lambda_j \models \lambda_i$, and hence $\lambda_i \equiv \lambda_j$. We assume without loss of generality that $i < j$. Now we know from previous paragraphs that **Π-Entail** gives the correct response when given conjuncts of ζ as input. It follows that we have **Π-Entail**(λ_i,λ_j)=**yes**. But that means that we will not enter the if-loop when examining λ_j, so λ'_j will not be a conjunct of ζ', contradicting our assumption to the contrary. Let us then consider the other alternative which is that $\lambda_j \not\models \lambda_i$. In this case, we have **Π-Entail**(λ_i,λ_j)=**yes** and **Π-Entail**(λ_i,λ_j)=**yes**. This means that we will not enter the if-loop on λ_j, contradicting the fact that λ'_j is a conjunct of ζ'. We have thus shown that there can be no unnecessary conjuncts in ζ'.

We will next prove that the conjuncts of ζ' satisfy the required properties. Let λ'_i be a conjunct of ζ', which is obtained from the conjunct λ_i of ζ via the

modifications in Step 2 of **BackToPINF**. We note that all modal disjuncts of λ_i' must be of the form $\Box_k$**BackToPINF**(ψ) where $\Box_k\psi$ is a disjunct of λ_i or $\Diamond_k$**BackToPINF**(ψ) where $\Diamond_k\psi$ is a disjunct of λ_i. We have seen earlier in the proof that if ψ is such that $\Box_k\psi$ or $\Diamond_k\psi$ is a disjunct of some conjunct of ζ, then ψ must be equal to Π-**LangInt**$(\gamma,\mathcal{L})$ for some γ such that $\Box_k\gamma$ or $\Diamond_k\gamma$ is a subformula of φ. This means that we can apply the induction hypothesis to all of the modal disjuncts of λ_i'. We find that if **BackToPINF**(ψ) is such that $\Box_k$**BackToPINF**(ψ) or $\Diamond_k$**BackToPINF**(ψ) is a disjunct of λ_i', then **BackToPINF**(ψ) is a formula in prime implicate normal form which is equivalent to ψ. It follows that λ_i' satisfies part 3(c)(iii) of Definition 6.2.1. We next note that since ζ has been shown to satisfy all right-hand-side conditions, which means that λ_i has at most one $\Diamond_k$-disjunct for each $1 \leq k \leq n$, and is such that if it has disjuncts $\Diamond_k\gamma$ and $\Box_k\mu$, then $\gamma \models \mu$. Since the modifications in Step 2 never modify the number of $\Diamond$-disjuncts, the former property must also be satisfied by λ_i'. The latter property must also be satisfied by λ_i' since we have seen that the modifications in Step 2 are equivalence-preserving with respect to each disjunct.

Finally, we must show that the conjuncts of λ_i' satisfies property 3(c)(i), i.e. it does not contain any unnecessary disjuncts. We known that λ_i, and hence λ_i', cannot contain any unnecessary propositional disjuncts, since this would imply that a conjunct of φ had unnecessary disjuncts, which contradicts the fact that φ is in prime implicate normal form. Now suppose that there is some disjunct $\Diamond_k\psi$ of λ_i' which implies some other disjunct of λ_i'. Now we know that $\Diamond_k\psi$ cannot be unsatisfiable, since that would imply the presence of an unsatisfiable disjunct in ζ, which cannot happen since ζ satisfies all left-hand-side conditions. But then according to Theorem 2.3.3, there must be a second $\Diamond_k$-disjunct of λ_i', which we have shown in the previous paragraph to be impossible. Thus, any unnecessary disjunct in λ_i' must be a $\Box$-formula. Let us then suppose that there are disjuncts $\Box_k\alpha_p$ and $\Box_k\alpha_q$ of λ_i' such that $\Box_k\alpha_p \models \Box_k\alpha_q$ (the disjuncts must have the same modal operator, otherwise the entailment wouldn't hold). We know from the previous paragraph that α_p and α_q are in prime implicate normal form. It follows then that Π-**Entail**(α_p, α_q) =**yes**. If $\alpha_q \not\models \alpha_p$, then we will also have Π-**Entail**(α_q, α_p) =**no**, which means that $\Box_k\alpha_p$ will be removed from λ_i' in Step 2(b), which contradicts the fact that it appears in ζ'. If instead we have $\alpha_q \models \alpha_p$, then we will have Π-**Entail**(α_q, α_p) =**yes**, and either $\Box_k\alpha_p$ or $\Box_k\alpha_q$ will be deleted from λ_i', which again is a contradiction.

We have thus proven that if ζ is such that $\zeta = \Pi$-**LangInt**$(\varphi, \mathcal{L})$ for some formula φ in prime implicate normal form and some signature $\mathcal{L}$, the output of **BackToPINF** on input ζ is a formula in prime implicate normal form which is

equivalent to ζ. As the algorithm Π-**LangInt** was shown in Lemma 6.3.21 to run in linear time (and hence space), all that remains to be shown is that the procedure **BackToPINF** terminates in polynomial time in the size of its input. We show by induction on the depth of the input formula that **BackToPINF** terminates in quadratic time. The base case is when the input formula ζ has depth 0. We trivially have the result since **BackToPINF** terminates in linear time when the input formula is propositional. Now suppose that **BackToPINF** terminates in quadratic time in the size of the input formula for formulae of depth at most d, and suppose ζ has depth $d+1$. We remark that evaluating the if-condition in Step 2 for the different values of i involves at most 3 calls to Π-**Entail** for each pair of indices $i \neq j$. Since Π-**Entail** terminates in $O(|\lambda_i|\,|\lambda_j|)$ steps on input (λ_i, λ_j) or (λ_j, λ_i), it follows that evaluating the if-condition in Step 2 takes in total $O(\Sigma_{i\neq j}|\lambda_i|\,|\lambda_j|)$ steps. In Step 2a, for each $1 \leq i \leq m$, and each ψ such that $\Box_k\psi$ or $\Diamond_k\psi$ is a disjunct of λ_i, we make a recursive call to **BackToPINF** on input ψ. By the induction hypothesis, we know that the call terminates in $O(|\psi|^2)$ steps. But since for each λ_i, the sum of the lengths of the different ψ is at most $|\lambda_i|$, it follows that Step 2a can be executed in $O(|\lambda_i|^2)$ steps. We also observe that the output of **BackToPINF** is always smaller or equal to its input, which means that λ_i' at the end of Step 2a is no larger than λ_i. This means that when comparing the disjuncts of λ_i' using Π-**Entail**, we need require only $\mathrm{O}(|\lambda_i|^2)$ steps. So the overall time spent on Step 2a and 2b is $O(\Sigma_{i=1}^m|\lambda_i|^2)$. It follows that the algorithm runs in $O(\Sigma_{i\neq j}|\lambda_i|\,|\lambda_j|) + O(\Sigma_{i=1}^m|\lambda_i|^2)$ steps, which is in $O(|\zeta|^2)$ since $\Sigma_{i=1}^m|\lambda_i| \leq |\zeta|$. We have thus shown how to generate in polynomial time for any formula in prime implicate normal form an $\mathcal{L}$-interpolant which is itself in prime implicate normal form. □

6.3.3 Canonicity

Another interesting property of prime implicate normal form in propositional logic is that it is unique up to reordering of conjuncts and disjuncts. This means that prime implicate normal form provides a canonical way of representing propositional formulae.

In this subsection, we will show that the same holds true for formulae in $\mathcal{K}_n$. In order to properly formalize what it means for two formulae to be the same up to reordering of conjuncts and disjuncts, we will require the following definitions.

Definition 6.3.26.
We will say that a formula ψ is *reachable via one step of reordering* from a formula φ, written $\varphi \hookrightarrow_o \psi$, just in the case that there is a subformula σ of the form $\rho_1 \oplus ... \oplus \rho_k$

(where $\oplus \in \{\wedge, \vee\}$) of φ and a permutation p of $\{1, ..., k\}$ such that substituting the formula $\rho_{p(1)} \oplus ... \oplus \rho_{p(k)}$ for one or more occurrences of the subformula σ in φ yields ψ.

Definition 6.3.27.
We will say that formulae ψ and φ are *identical modulo reordering*, written $\varphi \backsimeq_o \psi$, just in the case that there are a sequence of formulae $\zeta_0 = \varphi, \zeta_1, ..., \zeta_n = \psi$ such that $\zeta_i \hookrightarrow_o \zeta_{i+1}$ for all $0 \leq i < n$.

Example 6.3.28.
We have $a \wedge b \wedge \Box(a \vee c) \hookrightarrow_o a \wedge b \wedge \Box(c \vee a)$ since $a \vee c$ is a subformula of $a \wedge b \wedge \Box(a \vee c)$ of the form $\rho_1 \vee \rho_2$, the permutation $(2, 1)$ of $\{1, 2\}$ gives the formula $\rho_2 \vee \rho_1 = c \vee a$, and substituting $c \vee a$ for the unique occurrence of $a \vee c$ gives $a \wedge b \wedge \Box(c \vee a)$. We also have $a \wedge b \wedge \Box(c \vee a) \hookrightarrow_o b \wedge \Box(c \vee a) \wedge a$ since $a \wedge b \wedge \Box(c \vee a)$ is its own subformula, and by rearranging its conjuncts according to the permutation $(3, 1, 2)$ we obtain $b \wedge \Box(c \vee a) \wedge a$. It follows that $a \wedge b \wedge \Box(a \vee c) \backsimeq_o b \wedge \Box(c \vee a) \wedge a$.

Theorem 6.3.29.
If φ and ψ are formulae in prime implicate normal form such that $\varphi \equiv \psi$, then φ and ψ are identical modulo reordering.

Proof. Let φ and ψ be two formulae in prime implicate normal form such that $\varphi \equiv \psi$. The proof is by induction on the depth of φ. The base case is when $\delta(\varphi) = 0$. If φ is a tautology or a contradiction, the result clearly holds since then we have either $\psi = \varphi = \top$ or $\psi = \varphi = \bot$, and so $\varphi \backsimeq_o \psi$. If φ and ψ are neither tautologous nor a contradiction, then consider some conjunct λ_i of φ. Since $\varphi \equiv \psi$, we must have $\psi \models \lambda_i$, and so by the **Covering** property (Theorem 3.2.8), there must be some prime implicate of ψ which implies λ_i. But ψ is in prime implicate normal form, so each of its prime implicates is equivalent to one of its conjuncts, which means there must be some conjunct, call it $\lambda'_{p(i)}$, such that $\lambda'_{p(i)} \models \lambda_i$. As λ_i is a prime implicate of φ and hence also of ψ, we must also have $\lambda_i \models \lambda'_{p(i)}$, so $\lambda_i \equiv \lambda'_{p(i)}$.

We have just shown that for each conjunct λ_i of φ, there is some conjunct $\lambda'_{p(i)}$ of ψ such that $\lambda_i \equiv \lambda'_{p(i)}$. We know that there can be at most one, and hence exactly one, such conjunct of ψ since otherwise ψ would have a redundant conjunct, which is impossible given part 3(a) of the definition of prime implicate normal form (Definition 6.2.1). Moreover, we cannot have $p(i) = p(j)$ for $i \neq j$, since that would mean that φ would have redundant conjuncts, which cannot happen since it too is in prime implicate normal form. This means that φ and ψ have exactly the same number of conjuncts, say n, and the function p we have defined is in fact a permutation of $\{1, ..., n\}$.

We will now show that $\lambda_i \backsimeq_o \lambda'_{p(i)}$. Let θ_j be some disjunct of λ_i. As $\delta(\varphi) = 0$, θ_j must be a propositional literal, and since $\lambda_i \models \lambda'_{p(i)}$, we must also have $\theta_j \models \lambda'_{p(i)}$. We know $\lambda'_{p(i)}$ is non-tautologous, so by Theorem 2.3.3, θ_j must imply the propositional part of $\lambda'_{p(i)}$, and so must imply, and hence be equal to, some propositional disjunct, say $\theta_q(j)'$, of $\lambda'_{p(i)}$. But that means that the disjuncts of λ_i must be exactly the disjuncts of $\lambda'_{p(i)}$ since otherwise $\lambda'_{p(i)}$ would contain an unnecessary disjunct, which is forbidden by part 3(c)(i) of Definition 6.2.1. For the same reason, there can be no repeated disjuncts in $\lambda_{p(i)'}$. It follows that λ_i and $\lambda'_{p(i)}$ have the same number of disjuncts, say m. We have thus shown that there is a permutation q of $\{1, ..., m\}$ such that $\theta_j = \theta'_{q(j)}$ for all $1 \leq j \leq m$. It follows that $\lambda_i \hookrightarrow_o \lambda'_{p(i)}$, and hence $\lambda_i \backsimeq_o \lambda'_{p(i)}$.

We have thus demonstrated that $\lambda_i \backsimeq_o \lambda'_{p(i)}$ for every $1 \leq i \leq n$. That means that we can transform φ into ψ by first transforming each subformula λ_i into $\lambda'_{p(i)}$ and then applying the permutation p to the conjuncts of the resulting formula. It follows that $\varphi \backsimeq_o \psi$.

Now let us assume that the result holds whenever the first formula has depth at most d, and let φ be of depth $d + 1$. Using exactly the same reasoning as in the base case, we can show that φ and ψ have the same number of conjuncts, say n, and we can find a permutation p of $\{1, ..., n\}$ such that λ_i is equivalent to $\lambda'_{p(i)}$. We now wish to show that $\lambda_i \backsimeq_o \lambda'_{p(i)}$. Let θ_j be some disjunct of λ_i. If θ_j is a propositional literal, then we can apply the same reasoning as in the base to case to find some disjunct $\theta'_{q(j)}$ of $\lambda'_{p(i)}$ such that $\theta_j = \theta'_{q(j)}$. Suppose instead that θ_j is of the form $\Diamond_k \zeta$. We know that $\Diamond_k \zeta$ is not a contradiction, since otherwise λ_i would contain an unnecessary disjunct. Since $\lambda_i \models \lambda'_{p(i)}$, we must also have $\Diamond_k \zeta \models \lambda'_{p(i)}$. Then by Theorem 2.3.3, $\Diamond_k \zeta$ must imply the disjunction of the $\Diamond_k$-disjuncts of λ_i. But λ_i is in prime implicate normal form, so there can be only one such disjunct, call it $\theta'_{q(j)} = \Diamond_k \gamma$. We thus have $\zeta \models \gamma$. As $\lambda'_{p(i)} \models \lambda_i$, we must also have $\gamma \models \zeta$, and hence $\gamma \equiv \zeta$. Both ζ and γ are in prime implicate normal form and $\delta(\zeta) \leq d$, so the induction hypothesis applies, giving us $\zeta \backsimeq_o \gamma$. But that means that there is a sequence of reordering operations that transform ζ into γ, so by applying these same operations to the subformula ζ of $\Diamond_k \zeta$ we obtain the formula $\Diamond_k \gamma$, which means $\theta_j \backsimeq_o \theta'_{q(j)}$. Finally, consider the case where θ_j is of the form $\Box_k \zeta$. Then since $\lambda_i \models \lambda'_{p(i)}$, we also have $\Box_k \zeta \models \lambda'_{p(i)}$. By Theorem 2.3.3, there must be some $\Box_k$-disjunct $\theta'_{q(j)} = \Box_k \gamma$ of $\lambda'_{p(i)}$ such that $\zeta \models \gamma \vee \alpha_1 \vee ... \vee \alpha_r$, where $\Diamond_k \alpha_1$, ..., $\Diamond_k \alpha_r$ are the $\Diamond_k$-disjuncts of $\lambda'_{p(i)}$. As $\lambda'_{p(i)}$ is in prime implicate normal form, by part 3(c)iv of Definition 6.2.1, we have $\gamma \equiv \gamma \vee \alpha_1 \vee ... \vee \alpha_r$, and hence $\zeta \models \gamma$. We can show similarly that $\gamma \models \zeta$, and hence $\gamma \equiv \zeta$. Now ζ and γ are in prime implicate normal form, and the depth of ζ is no greater than d, so the induction

hypothesis is applicable, yielding $\zeta \backsimeq_o \gamma$. Thus, there is a sequence of reordering operations transforming ζ into γ, which means that we can turn $\Box_k \zeta$ into $\Box_k \gamma$ by applying these same operations to the subformula ζ. So we have $\theta_j \backsimeq_o \theta'_{q(j)}$.

We have shown in the previous paragraph that for every disjunct θ_j of λ_i there is some disjunct $\theta'_{q(j)}$ of $\lambda'_{p(i)}$ such that $\theta_j \backsimeq_o \theta'_{q(j)}$. We also need to show that the function q that we have defined is in fact a permutation. For this, we need to show that every disjunct of λ_i maps to a different disjunct of $\lambda'_{p(i)}$, and that every disjunct of $\lambda'_{p(i)}$ is paired with some disjunct of λ_i. For the former statement, we simply note that if two disjuncts of λ_i map to the same disjunct of $\lambda'_{p(i)}$, then λ_i contains an unnecessary disjunct, which is forbidden by the definition of prime implicate normal form. For the second statement, we remark that since $\lambda'_{p(i)} \models \lambda_i$, every disjunct of $\lambda'_{p(i)}$ entails some disjunct of λ_i, and hence is equivalent to some disjunct of λ_i. If $\lambda'_{p(i)}$ were to contain more disjuncts than λ_i, then it would contain some unnecessary disjunct, which we know not to be the case. Thus, our function q defines a permutation with the required properties from Definition 6.3.26, so we have $\lambda_i \backsimeq_o \lambda'_{p(i)}$.

To complete the proof, we remark that we can change φ into ψ by first transforming each subformula λ_i into $\lambda'_{p(i)}$ (which is possible since $\lambda_i \backsimeq_o \lambda'_{p(i)}$) and then applying the permutation p to the conjuncts of the resulting formula. We have thus shown φ and ψ to be identical modulo reordering. □

We can use Theorem 6.3.29 to show that formulae in prime implicate normal form have minimal signatures and depths.

Theorem 6.3.30.
Let φ be in prime implicate normal form, and let ψ be such that $\varphi \equiv \psi$. Then $sig(\varphi) \subseteq sig(\psi)$ and $\delta(\varphi) \leq \delta(\psi)$.

Proof. Let φ be in prime implicate normal form, and let ψ be such that $\varphi \equiv \psi$. We show below in Theorem 6.4.3 that there is a formula ψ' in prime implicate normal form such that $\psi' \equiv \psi$, $sig(\psi') \subseteq sig(\psi)$, and $\delta(\varphi) \leq \delta(\psi)$. According to Theorem 6.3.29, φ and ψ' are identical modulo reordering. In particular, this means that φ and ψ' have the same signature and depth. It follows then that $sig(\varphi) \subseteq sig(\psi)' \subseteq sig(\psi)$ and $\delta(\varphi) \leq \delta(\psi)' \leq \delta(\psi)$. □

6.4 Computing Prime Implicate Normal Form

We have seen in the last section that formulae in prime implicate normal form enjoy some nice properties, but in order to take advantage of them, we need a

method for putting formulae into prime implicate normal form.

We present in this section the algorithm **Pinf** which transforms a given formula into an equivalent formula in prime implicate normal form. The first step of our algorithm is to check whether the input formula is unsatisfiable or tautologous, in which case we return respectively $\bot$ or $\top$. For all other formulae, we continue on to Step 2, where we use **GenPI** to generate the set of prime implicates of the input formula, which we then modify in Step 3 so that they satisfy all the conditions of Definition 6.2.1. We first check to see whether there are multiple $\Diamond_i$-disjuncts, in which case we group them together into a single disjunct. Next we make sure that the formulae behind the $\Box_i$-modalities are in the proper form by disjoining them with the formula behind the single $\Diamond_i$-disjunct (if there is one). We then check if each of the disjuncts in the clause is necessary, and we remove all disjuncts which are found to be redundant. After that, we consider the formulae appearing behind the modalities, and we put each of them into prime implicate normal form. Finally, in Step 4, we return the conjunction of these modified prime implicates.

Algorithm 6.3 Pinf

Input: a formula φ

Output: a formula in prime implicate normal form equivalent to φ

(1) If **Sat**(φ)=**no**, return $\bot$. If **Entails**($\top, \varphi$)=**yes**, return $\top$.

(2) Set $\Sigma =$ **GenPI**(φ).

(3) For each π in Σ

(i) For each $1 \leq i \leq n$: if $Diam_i(\pi) = \{\psi_{i,1}, ..., \psi_{i,l_i}\}$ where $l_i > 1$, replace π by $\pi \setminus \{\Diamond_i\psi_{i,1}, ..., \Diamond_i\psi_{i,l_i}\} \cup \{\Diamond_i(\psi_{i,1} \vee ... \vee \psi_{i,l_i})\}$.

(ii) For each $1 \leq i \leq n$: if $Diam_i(\pi) = \{\epsilon\}$ and $Box_i(\pi) = \{\zeta_{i,1}, ...\zeta_{i,m_i}\}$, replace π by $\pi \setminus \{\Box_i\zeta_{i,1}, ..., \Box_i\zeta_{i,m_i}\} \cup \{\Box_i(\zeta_{i,1} \vee \epsilon), ..., \Box_i(\zeta_{i,m_i} \vee \epsilon)\}$.

(iii) For each disjunct μ in π: if $\pi \equiv \pi \setminus \{\mu\}$, replace π by $\pi \setminus \{\mu\}$.

(iv) For each $\psi \in \bigcup_{i=1}^{n}(Diam_i(\pi) \cup Box_i(\pi))$, replace ψ by **Pinf**(ψ).

(4) Return $\bigwedge_{\pi \in \Sigma} \pi$.

Example 6.4.1.

We use the algorithm **Pinf** to put the clauses from Example 6.2.2 into prime implicate normal form:

- **Pinf**($\Box_1 b \vee \Diamond_1 c$)=$\Box_1(b \vee c) \vee \Diamond_1 c$
- **Pinf**($\Diamond_1(a \wedge \Box_2\bot) \vee \Diamond_1(a \wedge \Box_2\top) \vee \neg c$)=$\Diamond_1 a \vee \neg c$, since **Pinf**($(a \wedge \Box_2\bot) \vee (a \wedge \Box_2\top)$)=$a$
- **Pinf**($\Diamond(a \wedge \neg a)$)=$\bot$ since $\Diamond(a \wedge \neg a) \models \bot$
- **Pinf**($\Box_1(a \vee \Box_2(b \vee \neg b))$)=$\top$ since $\models \Box_1(a \vee \Box_2(b \vee \neg b))$

- $\mathbf{Pinf}(a \vee \Box_1(a \wedge b) \vee \Box_1(a \wedge b \wedge \neg c)) = a \vee \Box_1(a \wedge b)$ since $\Box_1(a \wedge b \wedge \neg c)$ is unnecessary
- $\mathbf{Pinf}(\Box_1((a \wedge b) \vee c)) = \Box_1((a \vee c) \wedge (b \vee c))$ since $\mathbf{Pinf}((a \wedge b) \vee c) = (a \vee c) \wedge (b \vee c)$

Example 6.4.2.
We use the algorithm **Pinf** to put the formulae from Example 6.2.3 into prime implicate normal form:

- $\mathbf{Pinf}((a \vee \Diamond_2 c) \wedge (\neg a \vee c)) = (a \vee \Diamond_2 c) \wedge (\neg a \vee c) \wedge (c \vee \Diamond_2 c)$, since all conjuncts are in prime implicate normal form, every prime implicates is equivalent to some conjunct, and no conjunct is implied by another conjunct
- $\mathbf{Pinf}(a \wedge (a \vee \Box_3 b)) = a$, since a is the only prime implicate of $a \wedge (a \vee \Box_3 b)$, and a is in prime implicate normal form
- $\mathbf{Pinf}((a \vee \neg d) \wedge \Box_1((a \wedge b) \vee c)) = (a \vee \neg d) \wedge \Box_1((a \vee c) \wedge (b \vee c))$, since all conjuncts are in prime implicate normal form, every prime implicate is equivalent to some conjunct, and no conjunct is implied by another conjunct

The correctness of **Pinf** is shown in the next theorem.

Theorem 6.4.3.
The output of **Pinf** *is a formula in prime implicate normal form which is equivalent to the input formula, has a signature contained in the signature of the input formula, and has depth at most that of the input formula.*

Proof. The proof is by induction on the depth of the input formula φ. If φ has depth 0, then either $\varphi \models \bot$, or $\models \varphi$, or φ is neither unsatisfiable nor tautologous. In the first two cases, the result trivially holds. In the third case, we will continue on to Step 2 where we set Σ equal to the output of $\mathbf{GenPI}(\varphi)$. Because of Theorem 4.1.4 we know that every element in Σ is a prime implicate of φ and that all prime implicates of φ are equivalent to some element in Σ. It follows then from Theorem 3.2.9 that φ is equivalent to conjunction of the elements in Σ. Moreover, we know from Theorem 4.1.5 that the signatures of the formulae in Σ are all contained in the signature of φ and that the depths of the elements in Σ are bounded above by $\delta(\varphi)$. As φ is assumed to be propositional, the only modification we may make to Σ in Step 3 is to eliminate repeated literals appearing in the prime implicates. It follows then that the algorithm terminates and returns a formula in prime implicate normal form which is equivalent to φ, has a signature contained in $sig(\varphi)$, and has depth at most $\delta(\varphi)$.

Suppose next that the result holds whenever the input formula has depth at most k, and let φ be a formula of depth $k+1$. Clearly the result holds if $\varphi \models \bot$ or $\models \varphi$. Suppose then that φ is neither unsatisfiable nor tautologous. In Step 2,

we set Σ equal to the output of **GenPI**(φ). By Theorem 4.1.4, we know that the elements of Σ are precisely the prime implicates of φ, so φ must be equivalent to the conjunction of elements in Σ by Theorem 3.2.9. We also know from Theorem 4.1.5 that the signatures of the formulae in Σ are all contained in the signature of φ and that the depths of the elements in Σ cannot exceed $\delta(\varphi)$. Thus all we need to show is that the operations performed on the formulae in Σ in Step 3 are equivalence-, signature-, and depth-preserving. For (i) and (ii), this follows directly from Theorem 2.3.1, and for (iii), this is obvious. For (iv), this follows from the induction hypothesis since we apply the function **Pinf** to formulae with depth at most k. We have thus shown that the formula output by **Pinf**(φ) is equivalent to φ, has signature contained in $sig(\varphi)$, and depth at most $\delta(\varphi)$. We now verify that **Pinf**(φ) is in prime implicate normal form. Clearly, **Pinf**(φ) is a conjunction of clauses, since the elements in Σ are originally clauses, and the modifications in Step 3 do not change this. As we have shown the operations in Step 3 to be equivalence-preserving, it follows that the conjuncts of **Pinf**(φ) are all prime implicates of C and that each prime implicate of C is equivalent to some conjunct of **Pinf**(φ). Moreover, the conjuncts all satisfy the other conditions of Definition 6.2.1. We have $|Diam_i(\varphi)| \leq 1$ for every $1 \leq i \leq n$ because of part (i) of Step 3. Because of Step 3 (ii), we know that for if there are disjuncts $\Diamond_i\epsilon$ and $\Box_i\psi$, then $\epsilon \models \psi$. We also know that there are no redundant disjuncts since all unnecessary disjuncts were eliminated in Step 3 (iii). Finally, we can be sure that all of the formulae appearing behind the modal operators are in prime implicate normal form because of part (iv) of Step 3. We have thus shown that **Pinf**(φ) is in prime implicate normal form, completing the proof. □

6.5 Spatial Complexity of Prime Implicate Normal Form

In the current section, we investigate the spatial complexity of prime implicate normal form in order to determine how much more space is needed in the worst-case to represent a formula in prime implicate normal form.

It is well-known that in propositional logic the transformation to prime implicate normal form can result in an exponential blowup in the size of the formula (cf. [CM78]). The blowup can never be more than singly-exponential since there are at most 3^n distinct clauses on n variables.

Theorem 6.5.1.
Every propositional formula built from n propositional variables is equivalent to a formula in prime implicate normal form whose length is single exponential in n.

We now prove that for arbitrary formulae in $\mathcal{K}_n$ the transformation to prime implicate normal form involves an at most double exponential blowup in formula length.

Theorem 6.5.2.
Every formula φ in $\mathcal{K}_n$ is equivalent to a formula in prime implicate normal form whose length is at most double exponential in $|\varphi|$.

Proof. We assume throughout the proof that the input to **Pinf** is in NNF. This is without loss of generality since the transformation to NNF is linear (Theorem 2.4.2). We will use $f_l(k)$ to denote the maximal length of the output of **Pinf** when the input formula has depth k and l mutually non-equivalent literal subformulae. We know from Theorem 6.5.1 that there exists some polynomial q such that every propositional formula built using at most m propositional variables is equivalent to some propositional formula in prime implicate normal form with length at most $2^{q(m)}$. As the number of propositional variables appearing in a formula can never exceed the number of mutually non-equivalent literal subformulae appearing in the formula, it follows that there exists some polynomial function p such that $f_l(0) \leq 2^{p(l)}$.

Now that we have obtained an upper bound on $f_l(0)$, we try to obtain an upper bound on $f_l(k+1)$ in terms of $f_l(k)$. Consider some formula φ with depth $k+1$ and having at most l mutually non-equivalent literal subformulae. The output of $\mathbf{Pinf}(\varphi)$ is a conjunction of clauses, one for each prime implicate of φ. We know from the proof of Theorem 4.1.12 that there can be no more than l^{2^l} prime implicates of φ modulo equivalence. As the output of $\mathbf{Pinf}(\varphi)$ is in prime implicate normal form, and formulae in prime implicate normal form have one conjunct per equivalence class of prime implicates, there can be at most l^{2^l} conjuncts in the output of $\mathbf{Pinf}(\varphi)$.

We also know that every prime implicate of φ is equivalent to some clause having at most 2^l disjuncts (cf. proof of Theorem 4.1.6). We want to show that the elements in Σ at the beginning of Step 4 also have at most 2^l disjuncts each. Let us then consider some formula π which is a conjunct of $\mathbf{Pinf}(\varphi)$, and let π' be a clause with at most 2^l disjuncts which is equivalent to π. We will suppose that for any pair of disjuncts $\Box_i\zeta$ and $\Diamond_i\theta$ of π' we have $\theta \models \zeta$. This is without loss of generality since any clause can be transformed into an equivalent clause with the same number of disjuncts and satisfying this condition (cf. Theorem 2.3.1). As π is in prime implicate normal form (by correctness of **Pinf**, Theorem 6.4.3), it cannot have any unnecessary disjuncts, which means in particular that there can be no unsatisfiable disjuncts, nor any disjunct which implies another disjunct.

Since $\pi \models \pi'$, we know that $Prop(\pi) \subseteq Prop(\pi')$. As there can be no repeated propositional disjuncts in π, the number of propositional disjuncts in π' must be at least as great as the number of propositional disjuncts in π. Next suppose that π possesses a disjunct $\Diamond_i\psi$. We know that $\Diamond_i\psi$ is satisfiable, so by Theorem 2.3.3 there must be at least one $\Diamond_i$-disjunct in π'. As we know π to have at most one $\Diamond_i$-disjunct per i, it follows that π' has at least as many $\Diamond$-disjuncts as π. Finally, we want to show that number of $\Box$-disjuncts of π is bounded above by the number of $\Box$-disjuncts of π'. We first remark that if π contains a disjunct $\Box_i\chi$, then there must be some disjunct $\Box_i\zeta$ of π' such that $\chi \models \zeta$ (because of Theorem 2.3.3 and our assumptions on the structure of π'). We need to make sure however that each $\Box_i$-disjunct of π matches up with a different $\Box_i$-disjunct of π'. Let us suppose then that $\Box_i\chi_1$ and $\Box_i\chi_2$ are disjuncts of π which imply a single disjunct $\Box_i\zeta$ of π'. We thus have $\chi_1 \models \zeta$ and $\chi_2 \models \zeta$. As $\pi' \models \pi$, and π is in prime implicate normal form, we must have $\zeta \models \chi_j$ for some disjunct $\Box_i\chi_j$ of π. It follows that $\chi_1 \models \chi_j$ and $\chi_2 \models \chi_j$. If $j = 1$, then we have $\chi_2 \models \chi_1$, making the disjunct $\Box_i\chi_2$ unnecessary, contradicting our assumption that π is in prime implicate normal form. For other values of j, we obtain a contradiction in a similar manner. Thus we can conclude that there can be no pair of disjuncts $\Box_i\chi_1$ and $\Box_i\chi_2$ which imply the same disjunct of π'. It follows then that the total number of $\Box$-disjuncts in π' is at least as great as that of π. We have thus shown that π' has at least as many disjuncts as π, and hence that π has no more than 2^l disjuncts.

We now want to place a bound on the size of the disjuncts appearing in the conjuncts of $\mathbf{Pinf}(\varphi)$. Consider some conjunct π of $\mathbf{Pinf}(\varphi)$, and let λ be the element of $\mathbf{GenPI}(\varphi)$ which was transformed into π via the modifications in Step 3 of **Pinf**. Besides the propositional disjuncts which have length at most 2, there are two types of disjuncts which may appear in π: formulae of the form $\Diamond_i(\mathbf{Pinf}(\psi_1 \vee ... \vee \psi_r)$ where $Diam_i(\lambda) = \{\psi_1, ..., \psi_r\}$, and formulae of the form $\Box_i\mathbf{Pinf}(\epsilon \vee \psi_1 \vee ... \vee \psi_r)$ where $\epsilon \in Box_i(\lambda)$ and $Diam_i(\lambda) = \{\psi_1, ..., \psi_r\}$. Now we know from Theorem 4.1.5 that every literal subformula of one of the elements in $Diam_i(\lambda) \cup Box_i(\lambda)$ must also be a literal subformula of φ. That means that if $\epsilon \in Box_i(\lambda)$ and $Diam_i(\lambda) = \{\psi_1, ..., \psi_r\}$, then all the literal subformulae appearing in $\psi_1 \vee ... \vee \psi_r$ or $\epsilon \vee \psi_1 \vee ... \vee \psi_r$ also appear in φ. As we have assumed there to be at most l mutually non-equivalent literal subformulae in φ, it follows that there can be no more than l mutually non-equivalent literal subformulae in $\psi_1 \vee ... \vee \psi_r$ or $\epsilon \vee \psi_1 \vee ... \vee \psi_r$. We also know that the disjuncts of λ have depth at most $k+1$ (Theorem 4.1.5), which means that any formula of the form $\psi_1 \vee ... \vee \psi_r$ or $\epsilon \vee \psi_1 \vee ... \vee \psi_r$ where $\epsilon \in Box_i(\lambda)$ and $Diam_i(\lambda) = \{\psi_1, ..., \psi_r\}$ must have depth no greater than k. We can thus conclude that $|\mathbf{Pinf}(\psi_1 \vee ... \vee \psi_n)| \leq f_l(k)$ and $|\mathbf{Pinf}(\epsilon \vee \psi_1 \vee ... \vee \psi_r)| \leq f_l(k)$ for $\epsilon \in Box_i(\lambda)$

and $Diam_i(\lambda) = \{\psi_1, ..., \psi_r\}$, which means that any disjunct in λ must have length at most $f_l(k) + 1$ (the extra 1 is for the modality).

Putting all of this together, we obtain the following relationship between $f_l(k+1)$ and $f_l(k)$:

$$f_l(k+1) \leq l^{2^l}(2^l(f_l(k)+1)+1)$$

Here the l^{2^l} gives the maximal number of conjuncts, 2^l gives the maximal number of disjuncts per conjunct, $f_l(k) + 1$ gives the maximal size of the disjuncts, and the two extra 1's in the formula are for the $\wedge$ and $\vee$ symbols which connect the different conjuncts and disjuncts. Using standard techniques for solving first-order linear recurrence relations, we arrive at the following:

$$f_l(k) \in O((l^{2^l}2^l)^k f_l(0))$$

It is not hard to see that this expression is no more than double exponential in l. Now suppose that φ is a formula with l mutually non-equivalent literal subformulae and depth k. We know that the size of $\mathbf{Pinf}(\varphi)$ is bounded above by $f_l(k)$. As the number of literal subformulae in a formula φ can never exceed $|\varphi|$, we must have $l \leq |\varphi|$. We also know that the depth of φ is bounded by the length of φ, i.e. $k = \delta(\varphi) \leq |\varphi|$. This means that the above expression is at most double exponential in $|\varphi|$, so $|\mathbf{Pinf}(\varphi)|$ must also be at most than double-exponential in $|\varphi|$. □

We now prove this upper bound to be optimal by showing that in some cases the transformation to prime implicate normal form may involve a double exponential blowup in formula size.

Theorem 6.5.3.
There exist formulae φ such that the smallest equivalent formula in prime implicate normal form has length which is double exponential in the length of φ.

Proof. In Theorem 4.1.7 of Chapter 4, we exhibited a formula φ such that the number of non-equivalent prime implicates of φ was double exponential in $|\varphi|$. Any formula in prime implicate normal form which is equivalent to φ must have double-exponentially many conjuncts, and hence a length which is double exponential in $|\varphi|$. □

6.6 Related Work

Normal forms have been proposed for a number of description logics. Indeed, most of the subsumption algorithms that have been introduced for subpropositional

description logics involve a normalization step in which concepts are put into some type of normal form. This is the case for instance for the description logics $\mathcal{FL}_0$ [LB87], CLASSIC [BPS94], $\mathcal{ALN}$ [Mol98], and $\mathcal{ALE}$ [BKM99]. There has been relatively little work however on normal forms for modal logics or for more expressive description logics which support full disjunction. Two notable exceptions are the disjunctive form introduced for the mu-calculus in [JW95] and adapted to $\mathcal{ALC}$ in [tCCMV06] and the linkless normal form for $\mathcal{ALC}$ concepts recently proposed in [FO07]. Both of these normal forms give rise to corresponding normal forms for $\mathcal{K}_n$ formulae via the correspondence introduced in Chapter 2.

In this section, we examine some of the properties of disjunctive and linkless normal form and compare them to our own. One criteria in our evaluation will be the relative succinctness of the normal forms. We recall the formal definition of this notion:

Definition 6.6.1.
Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be sets of $\mathcal{K}_n$-formulae. We say that $\mathcal{L}_1$ is *at least as succinct* as $\mathcal{L}_2$, just in the case that there exists a polynomial function p such that for every formula $\varphi \in \mathcal{L}_2$ there exists a formula $\psi \in \mathcal{L}_1$ such that $\varphi \equiv \psi$ and $|\psi| \leq p(|\varphi|)$.

6.6.1 Disjunctive form

Disjunctive form was first introduced in [JW95] as a normal form for mu-calculus formulae. In more recent work [tCCMV06], B. ten Cate and colleagues have used disjunctive form as a normal form for concepts in $\mathcal{ALC}$. We rephrase their definition in terms of $\mathcal{K}_n$ formulae:

Definition 6.6.2 (Disjunctive Form).
If Ψ is a set of $\mathcal{K}_n$ formulae, then $\nabla_i \Psi$ stands for the formula:

$$\bigwedge_{\psi \in \Psi} \Diamond_i \psi \wedge \Box_i (\bigvee_{\psi \in \Psi} \psi)$$

In the limit case where $\Psi = \emptyset$, we have $\nabla_i \Psi = \Box_i \bot$ The set of $\mathcal{K}_n$ formulae in *disjunctive form* is generated by the following recursive definition:

$$\varphi ::= \top \mid \bot \mid \pi \wedge \nabla_{i_1} \Psi_1 \wedge ... \wedge \nabla_{i_k} \Psi_k \mid \varphi \vee \varphi$$

where π is a consistent conjunction of propositional literals, i_1, ..., i_k are distinct elements of $\{1, 2, ..., n\}$, and Φ_1, ..., Φ_k are finite sets of formulae in disjunctive form.

Disjunctive form can be seen as a description of a formula's models. Each of the disjuncts $\pi \wedge \nabla_{i_1} \Psi_1 \wedge ... \wedge \nabla_{i_k} \Psi_k$ represents a set of possible models in which the

root of model satisfies the partial valuation π, there is at least one i_j-successor satisfying each of the concepts in Ψ_{i_j}, and all i_j-successors satisfy at least one of the formulae in Ψ_{i_j}.

With regards to queries, satisfiability-testing of formulae in disjunctive form is easy (it is shown in [JW95] to be decidable in linear time), but tautology-testing, subsumption, and equivalence-testing cannot be carried out efficiently (unless P=NP):

Theorem 6.6.3.
*Deciding whether a formula in disjunctive form is a tautology is co*NP*-hard.*

Proof. Any propositional formula φ in DNF can be transformed in linear time into an equivalent formula φ' in disjunctive form by simply removing any unsatisfiable disjuncts from φ. This means that if we were able to test in polynomial time whether a formula in disjunctive form is a tautology, then we could do the same for propositional DNF formulae. As the DNF tautology problem is known to be co-NP-complete (cf. [GJ79]), it follows that testing whether a formula in disjunctive form is a tautology is a co-NP-hard problem. □

As both entailment and equivalence-testing can be used to identify tautologies, these tasks must also be co-NP-hard:

Corollary 6.6.4.
The entailment and equivalence problems for formulae in disjunctive form are both co-NP*-hard.*

The worst-case spatial complexity of disjunctive form is better than that of prime implicate normal form: every formula is equivalent to a formula in disjunctive form which is at most single-exponentially larger [tCCMV06]. It follows that there are $\mathcal{K}_n$-formulae which can be represented exponentially more compactly in disjunctive form than in prime implicate normal form.

Theorem 6.6.5.
Prime implicate normal form is not at least as succinct as disjunctive form.

Are there formulae which can be more compactly represented in prime implicate normal form than disjunctive form? The next theorem answers this question in the affirmative.

Lemma 6.6.6.
If φ is a $\mathcal{K}_n$-formula in disjunctive form which is equivalent to some satisfiable and non-tautologous propositional formula ψ, then there exists a propositional DNF formula $\varphi' \equiv \varphi$ with $|\varphi'| \leq |\varphi|$.

Proof. Let ψ be a satisfiable and non-tautologous propositional formula, and let $\varphi = \tau_1 \vee ... \vee \tau_k$ be a $\mathcal{K}_n$-formula in disjunctive form which is equivalent to ψ. Suppose furthermore that there is no shorter $\mathcal{K}_n$-formula in disjunctive form equivalent to ψ. It follows that φ cannot have any unsatisfiable disjuncts, else we could remove some disjuncts to find a shorter but equivalent formula. Now, each disjunct τ_i of φ must either be of the form π or $\pi \wedge \mu$, where π is a consistent conjunction of propositional literals and μ a conjunction of modal formulae. Define τ_i' to be the propositional part π of τ_i. Notice that $\varphi' = \tau_1' \vee ... \vee \tau_k'$ is a propositional formula in DNF which is the same length or shorter than φ. To complete the proof, we must show that φ' is equivalent to φ. The first direction ($\varphi \models \varphi'$) is obvious since $\tau_i \models \tau_i'$ for every i. For the second direction ($\varphi' \models \varphi$), let λ be some non-tautologous propositional clause implied by ψ. As $\varphi \equiv \psi$, it follows that each disjunct τ_i of φ implies λ, i.e. $\tau_i \wedge \neg\lambda \models \bot$. It follows from Theorem 2.3.3 and the fact that both $\tau_i \not\models \bot$ and $\not\models \lambda$ that one of the propositional literal disjuncts of λ is a conjunct of τ_i. But that means that the same propositional literal must appear as a conjunct of τ_i', so $\tau_i' \models \lambda$. We have thus shown that $\varphi' \models \psi$, and hence $\varphi' \models \varphi$, completing the proof. □

Theorem 6.6.7.
Disjunctive form is not at least as succinct as prime implicate normal form.

Proof. Consider the propositional formula $\psi = \bigwedge_{i=1}^{n}(a_{i,1} \vee a_{i,2})$, and let φ be a $\mathcal{K}_n$-formula in disjunctive form which is equivalent to φ. By Lemma 6.6.6, we know that there must be a propositional DNF formula φ' which is equivalent to φ (and hence to ψ) and such that $|\varphi'| \leq |\varphi|$. It was shown in [DM02] that every propositional formula in DNF which is equivalent to ψ must have size exponential in n, which means that φ', hence φ, must be exponentially larger than ψ. This is enough to show the result since ψ is clearly in prime implicate normal form.

6.6.2 Linkless normal form

In [FO07, FO08, FGO09], Furbach and Obermaier investigate how linkless normal form (cf. [MR93]) can be lifted from propositional logic to the description logic $\mathcal{ALC}$. They propose in [FO07] a definition of linkless normal form for $\mathcal{ALC}$ concepts, and then in [FO08, FGO09], they introduce the notion of a linkless graph for representing $\mathcal{ALC}$ concepts and TBoxes. In this subsection, we restrict our discussion to linkless normal form, as the notion of linkless graph defines a certain graph-based data structure rather than a subset of formulae, making it less amenable to comparison with the other two normal forms.

We recall here Furbach and Olbermaier's definition of linkless normal form (which we have appropriately rephrased in terms of $\mathcal{K}_n$):

Definition 6.6.8 (Path).
The set of *paths* of a formula in NNF is defined as follows: [5]

$paths(\bot) = \emptyset$
$paths(\top) = \{\emptyset\}$
$paths(l) = \{\{l\}\}$, if l is a literal other than $\top$ or $\bot$
$paths(\varphi_1 \wedge \varphi_2) = \{X \cup Y \mid X \in paths(\varphi_1), Y \in paths(\varphi_2)\}$
$paths(\varphi_1 \vee \varphi_2) = paths(\varphi_1) \cup paths(\varphi_2)$

Definition 6.6.9 (Link).
A *link* is either a *formula link* or a *modal link*:

- A *formula link* of φ is a pair of complementary propositional literals occurring in a path of φ
- A *modal link* of φ is a set $S = \{\Diamond_i\chi, \Box_i\psi_1, ..., \Box_i\psi_k\}$ occurring in a path of φ such that every path in $\chi \wedge \psi_1 \wedge ... \wedge \psi_k$ contains a propositional or modal link, and no proper subset of S satisfies this condition.

Definition 6.6.10 (Linkless normal form).
A formula φ is in *linkless normal form* if it is in NNF, none of its paths contains a link, and for each subformula of φ of the form $\Box_i\psi$ or $\Diamond_i\psi$ the formula ψ is also in linkless normal form.

Satisfiability-testing for linkless formulae can be accomplished in polynomial time [FO07], and it is conjectured that uniform interpolation may be tractable for linkless formulae. However, tautology, entailment, and equivalence problems can be shown to be intractable using the same arguments as for disjunctive form.

Theorem 6.6.11.
*The tautology, entailment, and equivalence problems for formulae in linkless normal form are all co-*NP*-hard.*

Proof. We remark that any propositional formula φ in DNF can be transformed in linear time into an equivalent formula φ' in linkless normal form by simply removing any unsatisfiable disjuncts. This means we can use the same proof as for disjunctive formulae (Theorem 6.6.3). □

5. The definition of paths for the symbols $\bot$ and $\top$ is not entirely clear in [FO07], so we adopt here the definition from [FO08, FGO09].

With regards to spatial complexity, it is stated in [FO07] that the transformation to linkless normal form induces an at most single-exponential blowup in formula length. This means that linkless normal form can yield exponentially smaller representations than prime implicate normal form in some cases.

Theorem 6.6.12.
Prime implicate normal form is not at least as succinct as linkless normal form.

We do not currently know whether or not linkless normal form is at least as succinct as prime implicate normal form (to our knowledge, the relative succinctness of these normal forms has not yet been established in the case of propositional logic). We can show however that linkless normal form is strictly more concise than disjunctive form.

Theorem 6.6.13.
Linkless normal form is at least as succinct as disjunctive form, but disjunctive form is not at least as succinct as linkless normal form.

Proof. For the first statement, we show how to transform in polynomial time formulae from disjunctive form into equivalent formulae in linkless normal form. Let φ be a formula in disjunctive form. First we remove all unsatisfiable disjuncts from φ, replacing φ by $\bot$ if no disjuncts remain. Then for each remaining disjunct of the form $\pi \wedge \nabla_{i_1}\Psi_1 \wedge ... \wedge \nabla_{i_k}.\Psi_k$, we apply the same procedure to the subformulae Ψ_1, ..., Ψ_k. This procedure is clearly equivalence-preserving, and it can be executed in quadratic time since there are at most linearly many subformulae to treat, and satisfiability of disjunctive formulae is feasible in linear time. We claim that the resulting formula is in linkless normal form. For propositional formulae, this is clear since we just have a propositional DNF with satisfiable disjuncts, and such a formula must be in linkless normal form. Now suppose that the claim holds for formulae of depth at most d, and consider some formula φ' of depth $d+1$ which was obtained from a formula φ in disjunctive form via the outlined procedure. Then φ' is a disjunction whose disjuncts are either $\top$ or formulae of the form $\pi \wedge \nabla_{i_1}\Psi_1 \wedge ... \wedge \nabla_{i_k}.\Psi_k$. It is easily verified that every non-empty path of φ is of the form

$$\Pi \cup \{\Diamond_{i_j}\psi \,|\, \psi \in \Psi_j \text{ for some } 1 \leq j \leq k\} \cup \{\Box_{i_j}(\bigvee_{\psi\in\Psi_j} \psi) \,|\, 1 \leq j \leq k\}$$

where $\Pi = \{l \,|\, l \text{ is a conjunct of } \pi\}$. Any such path cannot have a formula link, since π does not contain complementary literals, nor can it contain a modal link, as the formula $\chi \wedge (\bigvee_{\psi\in\Psi_j} \psi)$ $(\chi \in \Psi_j)$ must be satisfiable, otherwise φ' would

contain an unsatisfiable disjunct. We also know from the induction hypothesis that each formula $\psi \in \bigcup_{j=1}^{k} \Psi_{i_j}$ must be in linkless normal form. It follows that φ is in linkless normal form as well.

For the second statement, we use the fact that the parity function from propositional logic can be represented in polynomial space in DNNF [Dar99]. As every propositional DNNF formula is also in linkless normal form [MR03], it follows that the parity function has a polynomial-sized representation in linkless normal form. To conclude the proof, we use the fact that parity function is known not to be polynomially representable in propositional DNF (cf. [DM02]), together with Lemma 6.6.6, which tells us that any $\mathcal{K}_n$-formula in disjunctive form which is equivalent to the parity function must be equivalent to some propositional DNF formula of the same or shorter length. □

7

Conclusion

Summary of our results

Research on consequence finding has predominantly focused on propositional logic and first-order logic. However, for many applications in artificial intelligence, neither of these logics is a good fit: propositional logic lacks the necessary expressivity, while first-order logic, though greatly expressive, is undecidable. In such circumstances, modal and description logics often prove a better choice. This is why in this thesis, we proposed a study of consequence finding for the modal logic $\mathcal{K}_n$, a well-known modal logic with close ties to the description logic $\mathcal{ALC}$.

It is not immediately clear how the key notions of consequence finding, prime implicates and prime implicants, should be defined in $\mathcal{K}_n$. This is due to the fact that prime implicates and prime implicants are normally defined in terms of clauses and terms, notions which are not typically used in modal logic. Instead of arbitrarily selecting a definition of clauses and terms for $\mathcal{K}_n$, we considered several possible definitions, which we evaluated first with respect to properties of their respective notions of clauses and terms, and then a second time with respect to the properties of the notions of prime implicates and prime implicants that they induce.

Two of the definitions (**D1** and **D2**) proved too inexpressive, and three of the definitions (**D3a**, **D3b**, and **D5**) yielded notions of prime implicates/implicants with highly undesirable behavior. Thankfully, the remaining definition (**D4**) proved much better-behaved. Indeed, we were able to show that the notions of prime implicates and prime implicants induced by **D4** satisfy all of our desired properties. This was quite a positive result, since it was not entirely clear *a priori* whether such a well-behaved definition existed for $\mathcal{K}_n$. Indeed, for the standard notion of prime implicates in first-order logic, many of the desirable properties of propositional

prime implicates do not hold [Mar91b, Mar91a].

Having selected a suitable notion of prime implicate, we next turned our attention to the principal reasoning problem in consequence finding, which is prime implicate generation. To this end, we proposed an algorithm **GenPI** for generating prime implicates of $\mathcal{K}_n$ formulae. Like many propositional prime implicate generation algorithms, **GenPI** leverages the **Distribution** property, which relates the prime implicates of a disjunction to the prime implicates of its disjuncts. The algorithm **GenPI** is a fairly straightforward implementation of the **Distribution** property, which makes **GenPI** easy to understand and analyze but not especially efficient. This is why we proposed several different modifications to **GenPI** which can be used to render it more practicable.

An examination of the formulae output by our algorithm allowed us to place upper bounds on the size and number of prime implicates. For prime implicate size, we showed that the shortest clausal representation of a prime implicate of a formula is never more than single-exponentially larger than the formula. Concerning the number of prime implicates, we demonstrated that a formula can have no more than double-exponentially many mutually non-equivalent prime implicates. We proved these bounds optimal by exhibiting specific formulae having single-exponential-sized prime implicates or double-exponentially many prime implicates. A natural question is whether we might be able to improve these results by using some kind of approximation of prime implicates, like the weaker notions of prime implicate induced by definitions **D1** and **D2**. Surprisingly, we showed that this is not the case: for **D1** and even for the extremely inexpressive **D2**, our lower bounds on the size and number of prime implicates continue to hold.

We next considered the problem of prime implicate recognition, with a view towards improving the efficiency of our generation algorithm. The fact that prime implicates in $\mathcal{K}_n$ can be exponentially large might suggest that prime implicate recognition requires exponential space. Fortunately, however, we showed that this is not the case by exhibiting a polynomial-space algorithm **TestPI** for deciding prime implicate recognition. This allowed us to prove the prime implicate recognition task PSPACE-complete, and thus of the same complexity as entailment in $\mathcal{K}_n$.

The results mentioned so far concern only the standard notion of prime implicates, but in many applications, more refined variants are needed. We investigated two such variants: new prime implicates, which allow one to isolate the novel facts which can be derived upon arrival of new information, and signature-bounded prime implicates, which allow one to characterize the consequences of a formula which are built from a given signature. We showed that most results for standard prime implicates can be transferred or adapted to these variants. The main exception was the

complexity of recognizing signature-bounded prime implicates, which we showed to be co-NEXPTIME-hard (and thus most likely not feasible in polynomial space).

Once rephrased in terms of prime implicants, the preceding results can be applied to the problem of abduction in $\mathcal{K}_n$: our notion of (new and signature-bounded) prime implicants can be used to define abductive explanations in $\mathcal{K}_n$, and our prime implicate generation algorithm provides a means of producing all of the abductive explanations to a given abduction problem. The notion of term underlying our definition of abductive explanations is more expressive than that used in [CP95]. This means that we are able to find explanations which are overlooked by Cialdea Mayer & Pirri's method. For instance, if we look for an explanation of the observation c given the background information $\Box(a \vee b) \rightarrow c$, we obtain $\Box(a \vee b)$, whereas their framework yields $\Box a$ and $\Box b$. This is an argument in favor of our approach since generally in abduction one is looking to find the *weakest* conditions guaranteeing the truth of the observation given the background information. Also of interest are our results on the size and number of prime implicants, as these yield corresponding lower bounds on the size and number of abductive explanations. In particular, our results imply that the abductive explanations of Cialdea Mayer & Pirri can have exponential size and be doubly exponentially many in number in the worst case, and thus behave no better in these respects than the notion of abductive explanation induced by our preferred definition **D4**.

The final part of this thesis was concerned with the application of our notion of prime implicate to knowledge compilation. We began by showing that the most obvious way of defining prime implicate normal form for $\mathcal{K}_n$ formulae yielded a normal form with very poor computational properties. This led us to propose a more sophisticated definition of prime implicate normal form, in which additional restrictions are placed on the representation of prime implicates. We showed that entailment between formulae in our normal form can be decided in polynomial time using a simple structural comparison algorithm. We then strengthened this result by providing more general conditions on the input formulae which guarantee the correct functioning of our structural comparison algorithm. This allowed us to identify some syntactic classes of entailment queries that are tractable for formulae in prime implicate normal form. Beyond facilitating entailment queries, our normal form also simplifies the uniform interpolation transformation. Indeed, we showed this transformation to be feasible in polynomial-time for formulae in prime implicate normal form. Having established the interest of our normal form, we next considered the problem of putting formulae into prime implicate normal form. The algorithm **Pinf** which we proposed for this purpose induces an at most double-exponential blowup in formula size, which we showed to be optimal.

Perspectives

Alternative generation methods: The prime implicate generation algorithm we proposed in this thesis follows the distribution-based approach, so a natural question for future research would be to investigate the possibility of using a resolution-based procedure to generate prime implicates in $\mathcal{K}_n$. Such an investigation could prove useful in the development of more targeted algorithms for generating new and signature-bounded prime implicates, as many existing algorithms for restricted consequence finding in propositional and first-order logic are resolution-based.

Alternative knowledge compilation methods: In this work, we showed how prime implicate normal form could be generalized to the modal logic $\mathcal{K}_n$ while preserving many of the nice properties of the propositional case. As prime implicate normal form is just one of several methods used to compile formulae in propositional logic (cf. [DM02], [FM08]), it would be interesting to see whether other knowledge compilation methods can be suitably lifted from propositional logic to $\mathcal{K}_n$. It may also prove worthwhile to investigate the extension of approximate knowledge compilation methods (cf. e.g. [del95], [CD97]) from propositional logic to $\mathcal{K}_n$, since such methods typically exhibit better spatial complexities than exact compilation methods.

Consequence finding in other modal and description logics: In this thesis, we studied consequence finding in the modal logic $\mathcal{K}_n$, so an obvious direction for future work is the extension of our investigation to other modal and description logics. Particularly of interest are modal logics of knowledge and belief (e.g. $\mathcal{S}5_n$) and expressive description logics used for the semantic web (e.g. extensions of $\mathcal{ALC}$ by number restrictions, nominals, and/or inverse roles). It would also be interesting to extend our investigation of consequence finding to description logic knowledge bases, where we do not only have concept expressions in isolation, but instead axioms and assertions. Since reasoning with respect to knowledge bases is generally more complicated than with isolated concept expressions, it might prove best to start by studying description logics of lower complexity than $\mathcal{ALC}$. Good choices could be the DL-Lite [CDL$^+$07] and $\mathcal{EL}$ [BBL05] families of description logics, since these logics have nice computational properties yet are expressive enough for interesting applications (in conceptual data modelling and bio-medical ontologies respectively).

A

Complexity Theory

Computational complexity theory (cf. [Pap94]) studies the computational resources that are required to solve different problems. Most often the problems that are considered are *decision problems*, that is, problems for which the answer on any given input is either yes or no. An example decision problem is that of deciding whether a natural number is prime since for every number the response is either yes or no. A decision problem can be defined formally as a pair of sets S, S', where S is a set of *instances* (possible inputs) and $S' \subseteq S$ is the set of *positive instances*, i.e. those for which the answer is yes. For the prime number decision problem, S would be the set of natural numbers, and S' the set of prime numbers.

We say that an algorithm solves a decision problem if it outputs the correct answer on all inputs (such algorithms are termed *decision procedures*). Decision problems can be assigned to different complexity classes based on the amount of time and/or space that is required to solve them. The class P comprises all decision problems which can be solved in polynomial time (in the size of the input) by a deterministic Turing machine. Decision problems in P are said to be efficiently solvable or *tractable*.

Another important complexity class is NP which contains all decision problems which can be solved in polynomial time by a non-deterministic Turing machine. The class co-NP is defined to be the set of all decision problems whose complement belongs to NP, i.e. those decision problems which can be obtained from a decision problem in NP by swapping yes- and no-instances.

The complexity class BH_2 (or DP) is a combination of the classes NP and co-NP. Formally, we say that a decision problem D belongs to the class BH_2 just in the case there exists decision problems $D_1 \in$NP and $D_2 \in$co-NP such that the set

of positive instances of D is precisely the intersection of the positive instances of D_1 and the positive instances of D_2.

The class PSPACE (respectively EXPSPACE) is comprised of those problems which can be solved in polynomial (respectively single-exponential) space by a deterministic Turing machine. By allowing non-determinism, we obtain the classes NPSPACE and NEXPSPACE, and by taking the complement, we get the classes co-PSPACE and co-EXPSPACE. It is well-known that PSPACE=NPSPACE=co-PSPACE and EXSPACE=NEXPSPACE=co-EXPSPACE.

We can also define in a similar manner to above the classes EXPTIME (those decision problems solvable in single-exponential time by a deterministic Turing machine), NEXPTIME (those decision problems solvable in single-exponential time by a non-deterministic Turing machine), and co-NEXPTIME (those decision problems whose complement belongs to NEXPTIME).

The aforementioned complexity classes are known to be related in the following manner:

$$\text{P} \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \subseteq \text{NEXPTIME} \subseteq \text{EXPSPACE}$$

It is a longstanding open question whether P=NP or whether NP=co-NP, but it is generally believed that these classes are distinct. Likewise, it is conjectured that EXPTIME$\neq$NEXPTIME and NEXPTIME$\neq$co-NEXPTIME.

A key notion of complexity theory is that of hardness. Informally speaking, a problem P is hard for C just in the case that it is at least as difficult as any problem in C. More formally, a problem P is said to be *hard* for a complexity class C if for every problem Q in C there exists a polynomial-time translation f which transforms every instance I of Q into an instance $f(I)$ of P in such a way that I is a positive instance of Q just in the case that $f(I)$ is a positive instance of P. If a problem P is both a member of a complexity class C and C-hard, then P is said to be C-complete. Satisfiability of formulae in propositional logic is NP-complete, whereas the complementary problem, namely unsatisfiability, is co-NP-complete. For $\mathcal{K}_n$, both satisfiability and unsatisfiability are PSPACE-complete (refer to Chapter 2.5).

Bibliography

[ACG+06] Philippe Adjiman, Philippe Chatalic, François Goasdoué, Marie-Christine Rousset, and Laurent Simon, *Distributed reasoning in a peer-to-peer setting: Application to the semantic web*, Journal of Artificial Intelligence Research **25** (2006), 269–314.

[BBL05] Franz Baader, Sebastian Brandt, and Carsten Lutz, *Pushing the $\mathcal{EL}$ envelope*, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05), 2005, pp. 364–369.

[BdV01] Patrick Blackburn, Martin de Rijke, and Yde Venema, *Modal logic*, Cambridge University Press, 2001.

[BHQ08] Meghyn Bienvenu, Andreas Herzig, and Guilin Qi, *Prime implicate-based belief revision operators*, Proceedings of the Eighteenth European Conference on Artificial Intelligence (ECAI'08), 2008, pp. 741–742.

[Bie07a] Meghyn Bienvenu, *Consequence finding in $\mathcal{ALC}$*, Proceedings of the Twentieth International Workshop on Description Logics (DL'07), CEUR Workshop Proceedings, vol. 250, 2007.

[Bie07b] ———, *Prime implicates and prime implicants in modal logic*, Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI'07), 2007, pp. 397–384.

[Bie08a] ———, *Complexity of abduction in the $\mathcal{EL}$ family of lightweight description logics*, Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR'08), 2008, pp. 220–230.

[Bie08b] ———, *Prime implicate normal form for $\mathcal{ALC}$ concepts*, Proceedings of the Twenty-Third Conference on Artificial Intelligence (AAAI'08), 2008, pp. 412–417.

[Bie08c] ———, *Prime implicate normal form for $\mathcal{ALC}$ concepts*, Proceedings of the Twentieth International Workshop on Description Logics (DL'08), CEUR Workshop Proceedings, vol. 353, 2008.

[Bie09] ———, *Prime implicates and prime implicants: From propositional to modal logic*, Accepted for publication in the Journal of Artifical Intelligence Research (2009).

[Bíl07] Marta Bílkovà, *Uniform interpolation and propositional quantifiers in modal logics*, Studia Logica **85** (2007), no. 1, 1–31.

[Bit07] Guilherme Bittencourt, *Combining syntax and semantics through prime form representation*, Journal of Logic and Computation **18** (2007), no. 1, 13–33.

[BKM99] Franz Baader, Ralf Küsters, and Ralf Molitor, *Computing least common subsumers in description logics with existential restrictions*, Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99), 1999, pp. 96–103.

[BPS94] Alex Borgida and Peter Patel-Schneider, *A semantics and complete algorithm for subsumption in the CLASSIC description logic*, Journal of Artificial Intelligence Research **1** (1994), 277–308.

[BSVMH84] Robert K. Brayton, Alberto L. Sangiovanni-Vincentelli, Curtis T. McMullen, and Gary D. Hachtel, *Logic minimization algorithms for VLSI synthesis*, Kluwer, 1984.

[BT02] Sebastian Brandt and Anni-Yasmin Turhan, *An approach for optimized approximation*, Proceedings of the KI-2002 Workshop on Applications of Description Logics (KIDLWS'01), 2002.

[BV04] Stephen Brown and Zvonko Vranesic, *Fundamentals of digital logic with VHDL design*, 2nd ed., McGraw-Hill, 2004.

[Bv06] Patrick Blackburn and Johan van Benthem, *Handbook of modal logic*, ch. Modal Logic: A Semantic Perspective, pp. 1–84, Elsevier, 2006.

[BvW06] Patrick Blackburn, Johan van Benthem, and Frank Wolter (eds.), *Handbook of modal logic*, Elsevier, 2006.

[Cas96] Thierry Castell, *Computation of prime implicates and prime implicants by a variant of the Davis and Putnam procedure*, Proceedings of the Eighth International Conference on Tools with Artificial Intelligence (ICTAI'96), 1996, pp. 428–429.

[CD97] Marco Cadoli and Francesco M. Donini, *A survey on knowledge compilation*, AI Communications **10** (1997), no. 3-4, 137–150.

[CDL+07] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati, *Tractable reasoning and efficient query answering in description logics: The DL-Lite family*, Journal of Automated Reasoning **39** (2007), no. 3, 385–429.

[Che80] Brian Chellas, *Modal logic: an introduction*, Cambridge University Press, 1980.

[CK90] C. C. Chang and H. Jerome Keisler, *Model theory*, North Holland, 1990.

[CM78] Ashok Chandra and George Markowsky, *On the number of prime implicants*, Discrete Mathematics **24** (1978), 7–11.

[CP95] Marta Cialdea Mayer and Fiora Pirri, *Propositional abduction in modal logic*, Logic Journal of the IGPL **3** (1995), no. 6, 907–919.

[Cra57] William Craig, *Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory*, Journal of Symbolic Logic **22** (1957), no. 3, 269–285.

[Dar99] Adnan Darwiche, *Compiling knowledge into decomposable negation normal form*, Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99), 1999, pp. 1096–1101.

[del95] Alvaro del Val, *An analysis of approximate knowledge compilation*, Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95), 1995, pp. 830–836.

[del99] ———, *A new method for consequence finding and compilation in restricted languages*, Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI'99), 1999, pp. 259–264.

[dK92] Johan de Kleer, *An improved incremental algorithm for generating prime implicates*, Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92), 1992, pp. 780–785.

[DLN+92] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Bernhard Hollunder, Werner Nutt, and Alberto Marchetti Spaccamela, *The complexity of existential qualification in concept languages*, Artificial Intelligence **53** (1992), 309–327.

[DM02] Adnan Darwiche and Pierre Marquis, *A knowledge compilation map*, Journal of Artificial Intelligence Research **17** (2002), 229–264.

[Don03] Francesco M. Donini, *The description logic handbook*, ch. Complexity of Reasoning, Cambridge University Press, 2003.

[DTR06] Chan Le Duc, Nhan Le Thanh, and Marie-Christine Rousset, *A compact representation for least common subsumers in the description logic $\mathcal{ALE}$*, AI Communications **19** (2006), no. 3, 239–273.

[EF89] Patrice Enjalbert and Luis Fariñas del Cerro, *Modal resolution in clausal form*, Theoretical Computer Science **65** (1989), no. 1, 1–33.

[EG95] Thomas Eiter and Georg Gottlob, *The complexity of logic-based abduction*, Journal of the ACM **42** (1995), no. 1, 3–42.

[FGO09] Ulrich Furbach, Heiko Gunther, and Claudia Obermaier, *A knowledge compilation technique for $\mathcal{ALC}$ TBoxes*, Proceedings of the 22th International Florida Artificial Intelligence Research Society Conference 2009 (FLAIRS'09), 2009.

[FM08] Hélène Fargier and Pierre Marquis, *Extending the knowledge compilation map: Krom, horn, affine and beyond*, Proceedings of the Twenty-Third Conference on Artificial Intelligence (AAAI'08), 2008, pp. 442–447.

[FO07] Ulrich Furbach and Claudia Obermaier, *Knowledge compilation for description logics*, Proceedings of the 3rd Workshop on Knowledge Engineering and Software Engineering (KESE), 2007.

[FO08] ———, *Precompiling $\mathcal{ALC}$ Tboxes and query answering*, Proceedings of the Fourth Workshop on Contexts and Ontologies, 2008.

[Ghi95] Silvio Ghilardi, *An algebraic theory of normal forms*, Annals of Pure and Applied Logic **71** (1995), no. 3, 189–245.

[GJ79] Michael R. Garey and David S. Johnson, *Computers and intractability. A guide to the theory of NP-completeness*, W. H. Freeman, 1979.

[GLW06] Silvio Ghilardi, Carsten Lutz, and Frank Wolter, *Did I damage my ontology? A case for conservative extensions in description logics*, Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR'06), 2006, pp. 187–197.

[GLWZ06] Silvio Ghilardi, Carsten Lutz, Frank Wolter, and Michael Zakharyaschev, *Conservative extensions in modal logic*, Proceedings of Sixth International Conference on Advances in Modal Logic (AiML06), 2006, pp. 187–207.

[GS96] Fausto Giunchiglia and Roberto Sebastiani, *A SAT-based decision procedure for $\mathcal{ALC}$*, Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96), 1996, pp. 304–314.

[GZ95] Silvio Ghilardi and Marek W. Zawadowski, *Undefinability of propositional quantifiers in the modal system S4*, Studia Logica **55** (1995), no. 2, 259–271.

[Hen63] Leon Henkin, *An extension of the Craig-Lyndon interpolation theorem*, Journal of Symbolic Logic **28** (1963), no. 3, 201–216.

[HHSS06] Ian Horrocks, Ullrich Hustadt, Ulrike Sattler, and Renate Schmidt, *Handbook of modal logic*, ch. Computational Modal Logic, pp. 181–248, Elsevier, 2006.

[HM08] Andreas Herzig and Jérôme Mengin, *Uniform interpolation by resolution in modal logic*, Proceedings of the Eleventh European Conference on Logics in Artificial Intelligence (JELIA'08), 2008, pp. 219–231.

[Ino92] Katsumo Inoue, *Linear resolution in consequence finding*, Artificial Intelligence **56** (1992), no. 2-3, 301–353.

[Jac92] Peter Jackson, *Computing prime implicates incrementally*, Proceedings of the Eleventh International Conference on Automated Deduction (CADE'92), 1992, pp. 253–267.

[JW95] David Janin and Igor Walukiewicz, *Automata for the modal mu-calculus and related results*, Proceedings of the Twentieth International Symposium on the Mathematical Foundations of Computer Science (MFCS'95), Lecture Notes in Computer Science, vol. 969, Springer, 1995, pp. 552–562.

[KT90] Alex Kean and George K. Tsiknis, *An incremental method for generating prime implicants/impicates*, Journal of Symbolic Computation **9** (1990), no. 2, 185–206.

[KWW08] Boris Konev, Dirk Walther, and Frank Wolter, *The logical difference problem for description logic terminologies*, Proceedings of the Fourth International Joint Conference on Automated Reasoning (IJCAR'08), 2008, pp. 259–274.

[Lad77] Richard Ladner, *The computational complexity of provability in systems of modal propositional logic*, SIAM Journal of Computing **6** (1977), no. 3, 467–480.

[Lak95] Gerhard Lakemeyer, *A logical account of relevance*, Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95), 1995, pp. 853–861.

[LB87] Hector J. Levesque and Ronald Brachman, *Expressiveness and tractability in knowledge representation and reasoning*, Computational Intelligence **3** (1987), 78–93.

[LLM03] Jérôme Lang, Paolo Liberatore, and Pierre Marquis, *Propositional independence: Formula-variable independence and forgetting*, Journal of Artificial Intelligence Research **18** (2003), 391–443.

[LR94] Fangzhen Lin and Raymond Reiter, *Forget it!*, AAAI Fall Symposium on Relevance, 1994, pp. 154–159.

[Mar91a] P. Marquis, *Contribution á l'étude des méthodes de construction d'hypothèses en intelligence artificielle*, In french, Université de Nancy I, 1991.

[Mar91b] Pierre Marquis, *Extending abduction from propositional to first-order logic*, Proceedings of Fundamentals of Artificial Intelligence Research Workshop, 1991, pp. 141–155.

[Mar00] ———, *Handbook on defeasible reasoning and uncertainty management systems*, vol. 5, ch. Consequence Finding Algorithms, pp. 41–145, Kluwer, 2000.

[McC56] Edward McCluskey, *Minimization of boolean functions*, Bell System Technical Journal **35** (1956), no. 6, 1417–1444.

[Mol98] Ralf Molitor, *Structural subsumption for $\mathcal{ALN}$*, LTCS-Report 98-03, RWTH Aachen, 1998.

[MR93] Neil V. Murray and Erik Rosenthal, *Dissolution: Making paths vanish*, Journal of the ACM **40** (1993), no. 3, 504–535.

[MR03] ———, *Tableaux, path dissolution, and decomposable negation normal form for knowledge compilation*, Proceedings of the International Conference on Analytic Tableaux and Related Methods (TABLEAUX), 2003, pp. 165–180.

[Nga93] Teow-Hin Ngair, *A new algorithm for incremental prime implicate generation*, Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI'93), 1993, pp. 46–51.

[Pag06] Maurice Pagnucco, *Knowledge compilation for belief change*, Proceedings of the Nineteenth Australian Conference on Artificial Intelligence (AI'06), 2006, pp. 90–99.

[Pap94] Christos Papadimitriou, *Computational complexity*, Addison Welsey, 1994.

[Prz89] Teodor C. Przymusinski, *An algorithm to compute circumscription*, Artificial Intelligence **38** (1989), no. 1, 49–73.

[Qui52] Willard V. Quine, *The problem of simplifying truth functions*, American Mathematical Monthly **59** (1952), no. 8, 521–531.

[Qui55] ———, *A way to simplify truth functions*, American Mathematical Monthly **62** (1955), no. 9, 627–631.

[RBM97] Anavai Ramesh, George Becker, and Neil V. Murray, *CNF and DNF considered harmful for computing prime implicants/implicates*, Journal of Automated Reasoning **18** (1997), no. 3, 337–356.

[Sch91] Klaus Schild, *A correspondence theory for terminological logics: Preliminary report*, Proceedings of the Twelth International Joint Conference on Artificial Intelligence (IJCAI'91), 1991, pp. 466–471.

[SCL69] James R. Slagle, Chin-Liang Chang, and Richard C. T. Lee, *Completeness theorems for semantic resolution in consequence-finding*, Proceedings of the First International Joint Conference on Artificial Intelligence (IJCAI), 1969, pp. 281–286.

[SdV01] Laurent Simon and Alvaro del Val, *Efficient consequence finding*, Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01), 2001, pp. 359–370.

[SL96] Bart Selman and Hector J. Levesque, *Support set selection for abductive and default reasoning*, Artificial Intelligence **82** (1996), 259–272.

[SM73] Larry J. Stockmeyer and Albert R. Meyer, *Word problems requiring exponential time: Preliminary report*, Proceedings of Fifth Annual ACM Symposium on Theory of Computing (STOC'73), 1973, pp. 1–9.

[Soc91] Rolf Socher, *Optimizing the clausal normal form transformation*, Journal of Automated Reasoning **7** (1991), no. 3, 325–336.

[SP99] A. K. Shiny and Arun K. Pujari, *An efficient algorithm to generate prime implicants*, Journal of Automated Reasoning **22** (1999), no. 2, 149–170.

[SSS91] Manfred Schmidt-Schauß and Gert Smolka, *Attributive concept descriptions with complements*, Artificial Intelligence **48** (1991), no. 1, 1–26.

[TB07] Anni-Yasmin Turhan and Yusri Bong, *Speeding up approximation with nicer concepts*, Proceedings of the Twentieth International Description Logic Workshop (DL'07), 2007.

[tCCMV06] Balder ten Cate, Willem Conradie, Martin Marx, and Yde Venema, *Definitorially complete description logics*, Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR'06), AAAI Press, 2006, pp. 79–89.

[Tis67] Pierre Tison, *Generalization of consensus theory and application to the minimization of boolean functions*, IEEE Transactions on Computers **C-16** (1967), 446–456.

[van83] Johan van Benthem, *Modal logic and classical logic*, Bibliopolis, 1983.

[Vis96] Albert Visser, *Gödel 96*, Lecture Notes in Logic, vol. 6, ch. Uniform interpolation and layered bisimulation, pp. 139–164, Springer-Verlag, 1996.

[You67] Daniel H. Younger, *Recognition and parsing of context-free languages in time n^3*, Information and Control **10** (1967), no. 2, 189–208.

Index

Consequence finding in modal logic

Meghyn GARNER BIENVENU

The key notion in consequence finding is that of prime implicates, which are defined to be the logically strongest clausal consequences of a formula. Prime implicates have proven useful in artificial intelligence, especially in knowledge compilation and abductive reasoning. In this thesis, we extend the investigation of prime implicates from propositional logic to the basic multi-modal logic K_n. We begin by comparing the properties of several plausible definitions of prime implicates in K_n in order to isolate the most suitable definition. We next study the computational aspects of the selected definition. Specifically, we provide algorithms for prime implicate generation and recognition, and we study the complexity of these tasks. Finally, we show how our notion of prime implicates can be used to define a normal form for K_n with interesting knowledge compilation properties.

Keywords : modal logic, automated reasoning, consequence finding, knowledge compilation

This thesis, presented and defended in Toulouse on May 7th, 2009, was carried out under the direction of Andreas Herzig. The author received the degree of Docteur en Informatique de l'Universite de Toulouse.

Universite de Toulouse - Institut de Recherche en Informatique de Toulouse
118 route de Narbonne, 31062 TOULOUSE Cedex 9, France