On Hierarchical Task Networks

Andreas Herzig[1], Laurent Perrussel[1], Zhanhao Xiao[1,2]

[1] University of Toulouse, IRIT, France
[2] Dept. of CS, Western Sydney University, Australia

Abstract. In planning based on hierarchical task networks (HTN), plans are generated by refining high-level actions ('compound tasks') into lower-level actions, until primitive actions are obtained that can be sent to execution. While a primitive action is defined by its precondition and effects, a high-level action is defined by zero, one or several methods: sets of (high-level or primitive) actions decomposing it together with a constraint. We give a semantics of HTNs in terms of dynamic logic with program inclusion. We propose postulates guaranteeing soundness and completeness of action refinement. We also show that hybrid planning can be analysed in the same dynamic logic framework.

1 Introduction

The two main approaches to deterministic AI planning are classical state-based planning [13] and Hierarchical Task Network (HTN) planning [5]. The former is based on action preconditions and effects. The latter is based on domain-specific heuristics about the decomposition of high-level actions ('compound tasks') into lower-level actions, until primitive actions ('primitive tasks') are obtained. It has no generally agreed semantics [7]. We here propose a semantics in terms of an extension of Propositional Dynamic Logic PDL [8] by a program inclusion operator. This framework sheds light on a problem that had not been investigated before: the soundness of HTN domain descriptions.

Let us illustrate HTNs and the soundness issue by an abstract example. Suppose the only method for high-level action α is $\langle \alpha, \langle \{(\beta, t)\}, (t, p)\rangle\rangle$. The couple $\langle \{(\beta, t)\}, (t, p)\rangle$ is a *task network*: (β, t) instantiates the action β by the temporal label t, and the constraint (t, p) stipulates that p should be true immediately after t. So the only way to perform α is by performing β, with postcondition p. Suppose moreover that β is also a high-level action and that its only method is $\langle \beta, \langle \{(b, t')\}, (t', \neg p)\rangle\rangle$. So the only way to perform β is to apply b, with postcondition $\neg p$. No task involving α can ever be solved, and we call such an HTN domain description *unsound*. It is reasonable to expect HTN domain descriptions not to contain unsound methods. This is a simple example, and more complex unsound methods can be designed. In this paper we show that PDL provides a framework where we can characterise sound domain descriptions. The PDL semantics also allows us to study whether the set of methods for a high-level action α is *complete*, in the sense that when the precondition of α is true then there is a method for α that is executable.

Beyond traditional HTN planning, we can show that PDL with program inclusion also provides a semantics for so-called hybrid planning. There, domain descriptions have preconditions and effects not only for primitive actions, but also for high-level actions. Following [11, 12], we consider that the effect of a high-level action is its main, *primary* effect. Indeed, it is not obvious to describe the effects of a high-level action α exhaustively. One of the reasons is that these effects depend on the way α is refined. For example, consider the high-level action of building a house. While its primary effect is that I have a house, its side effects depend on whether I build the house myself or hire a builder: I either have a bad back, or an empty bank account. We therefore consider that non-primitive actions are not described by their effects but only by their postconditions.

Our paper is organised as follows. In Section 2 we define PDL. In Section 3 we define HTN planning domains in PDL. In Section 4 we propose postulates of soundness, completeness and modularity. Section 5 concludes.[3]

2 PDL with inclusion of programs

We define syntax and semantics of a version of Propositional Dynamic Logic PDL having intersection and inclusion of programs and, for simplicity, with only boolean tests. Let $\mathtt{Prp}$ be a finite set of propositional variables, with typical elements p, q,... The set of boolean formulas built from $\mathtt{Prp}$ is noted $\mathtt{Fml}_{\mathsf{bool}}$. Let $\mathtt{Act}$ be a finite set of actions, with typical elements α, β,... In examples we use capital letters for propositional variables (such as HasHouse) and small letters for actions (such as buildHouse).

The set of programs $\mathtt{Pgm}_{\mathsf{PDL}}$ is defined by the following grammar:

$$\pi ::= \alpha \mid \pi;\pi \mid \pi \sqcup \pi \mid \pi \sqcap \pi \mid \pi^* \mid \varphi_0?$$

where $\alpha \in \mathtt{Act}$ and $\varphi_0 \in \mathtt{Fml}_{\mathsf{bool}}$. The program operators ";", "$\sqcup$", and "$\sqcap$" are sequential, nondeterministic and parallel composition, "*" is bounded iteration, and "?" is test. The set of formulas $\mathtt{Fml}_{\mathsf{PDL}}$ is defined by:

$$\varphi ::= p \mid \bot \mid \varphi \to \varphi \mid \langle\pi\rangle\varphi \mid \pi \sqsubseteq \pi$$

$\langle\pi\rangle\varphi$ reads "there is a possible execution of π after which φ is true" and $\pi' \sqsubseteq \pi$ reads "every execution of π' is also an execution of π". Subsets of $\mathtt{Fml}_{\mathsf{PDL}}$ are called *theories*. As usual, $[\pi]\varphi$ abbreviates $\neg\langle\pi\rangle\neg\varphi$.

A model is a triple $M = \langle W, R, V\rangle$ where W is a non-empty set of possible worlds, $R : \mathtt{Pgm}_{\mathsf{PDL}} \longrightarrow 2^{W\times W}$ associates accessibility relations R_π to programs, and $V : \mathtt{Prp} \longrightarrow 2^W$ is a valuation. The function R must satisfy some constraints:

$$\begin{aligned} R_{\pi_1;\pi_2} &= R_{\pi_1} \circ R_{\pi_2} & \qquad R_{\pi^*} &= (R_\pi)^* \\ R_{\pi_1\sqcup\pi_2} &= R_{\pi_1} \cup R_{\pi_2} & \qquad R_{\varphi_0?} &= \{\langle w, w\rangle \ : \ M, w \Vdash \varphi_0\} \\ R_{\pi_1\sqcap\pi_2} &= R_{\pi_1} \cap R_{\pi_2} \end{aligned}$$

[3] Our work is supported by CSC and CIMI. Thanks are due to the JELIA 2016 reviewers for their thorough comments. A long version of the paper with formal results and proofs is at www.irit.fr/~Andreas.Herzig/P/Jelia16htn.html.

Letting $R_\pi(w) = \{v \ : \ \langle w, v\rangle \in R_\pi\}$, the truth conditions for formulas are:

$M, w \Vdash p$ iff $w \in V(p)$
$M, w \nVdash \bot$
$M, w \Vdash \varphi \to \varphi'$ iff $M, w \nVdash \varphi$ or $M, w \Vdash \varphi'$
$M, w \Vdash \langle\pi\rangle\varphi$ iff $M, v \Vdash \varphi$ for some $v \in R_\pi(w)$
$M, w \Vdash \pi \sqsubseteq \pi'$ iff $R_\pi(w) \subseteq R_{\pi'}(w)$

For $\Gamma \subseteq \mathtt{Fml}_{\mathsf{PDL}}$, we define $\Gamma \models \varphi$ as: for every model M, if $M \Vdash \psi$ for every $\psi \in \Gamma$ then $M \Vdash \varphi$, where $M \Vdash \varphi$ stands for: $M, w \Vdash \varphi$ for all $w \in W$.

3 HTN planning in the PDL framework

HTN planning presupposes that the set of actions $\mathtt{Act}$ is partitioned into two sets: the set of primitive actions $\mathtt{Act}_0$ and the set of high-level actions $\mathtt{Act} \setminus \mathtt{Act}_0$. We use $a, b, \ldots$ for typical elements of $\mathtt{Act}_0$ (and, as before, $\alpha, \beta, \ldots$ for arbitrary elements of $\mathtt{Act}$). A *primitive plan* is a sequence of primitive actions. A *primitive program* is a program where only elements of $\mathtt{Act}_0$ occur.

We suppose that all actions have pre- and postconditions. The postconditions of primitive actions describe STRIPS-like effects in terms of add- and delete-lists. Non-primitive actions can have arbitrary boolean formulas as an postconditions. For example, the high-level action of leaving France may have postcondition $\neg\mathsf{InFrance} \wedge (\mathsf{InGermany} \vee \mathsf{InChina} \vee \ldots)$. In traditional HTNs, high-level actions have no postcondition, which can be captured by setting them to $\top$.

3.1 HTN planning domains

An *HTN planning domain* is a couple $\mathcal{D}_{\mathsf{htn}} = \langle \mathsf{Pre}, \mathsf{Post}, \mathsf{Ref}\rangle$ where $\mathsf{Pre}, \mathsf{Post} : \mathtt{Act} \longrightarrow \mathtt{Fml}_{\mathsf{bool}}$ and $\mathsf{Ref} : \mathtt{Act} \longrightarrow 2^{\mathtt{Pgm}_{\mathsf{PDL}}}$ such that for every $a \in \mathtt{Act}_0$, $\mathsf{Ref}(a) = \emptyset$ and $\mathsf{Post}(a)$ is of the form $\big(\bigwedge_{p\in\mathsf{eff}^+(a)} p\big) \wedge \big(\bigwedge_{p\in\mathsf{eff}^-(a)} \neg p\big)$, for some $\mathsf{eff}^+(a)$ and $\mathsf{eff}^-(\alpha)$ such that $\mathsf{eff}^+(a) \cap \mathsf{eff}^-(a) = \emptyset$. The refinement function Ref associates to each α its methods: the set of programs refining α. For the introductory example we have $\mathsf{Ref}(\alpha) = \{(\beta; p?)\}$, $\mathsf{Ref}(\beta) = \{(b; \neg p?)\}$, $\mathsf{Ref}(b) = \emptyset$, and, say, that all pre- and postconditions equal $\top$, except that $\mathsf{Post}(b) = \neg p$.

Example 1. An domain that can be found in almost all papers on HTN is that of an agent travelling from A to B:

$\mathsf{Pre}(\mathsf{goAB}) = \mathsf{AtA}$ $\quad \mathsf{Post}(\mathsf{goAB}) = \mathsf{AtB}$ $\quad \mathsf{Ref}(\mathsf{goAB}) = \{\mathsf{taxiAB}, \mathsf{walkAB}\}$
$\mathsf{Pre}(\mathsf{taxiAB}) = \mathsf{AtA}$ $\quad \mathsf{Post}(\mathsf{taxiAB}) = \mathsf{AtB}$ $\quad \mathsf{Ref}(\mathsf{taxiAB}) = \{(\mathsf{rideAB}; \mathsf{pay})\}$
$\mathsf{Pre}(\mathsf{walkAB}) = \mathsf{AtA}$ $\quad \mathsf{Post}(\mathsf{walkAB}) = \mathsf{AtB} \wedge \neg\mathsf{AtA}$ $\quad \mathsf{Ref}(\mathsf{walkAB}) = \emptyset$
$\mathsf{Pre}(\mathsf{rideAB}) = \mathsf{AtA}$ $\quad \mathsf{Post}(\mathsf{rideAB}) = \mathsf{AtB} \wedge \neg\mathsf{AtA}$ $\quad \mathsf{Ref}(\mathsf{rideAB}) = \emptyset$
$\mathsf{Pre}(\mathsf{pay}) = \mathsf{Money}$ $\quad \mathsf{Post}(\mathsf{pay}) = \neg\mathsf{Money}$ $\quad \mathsf{Ref}(\mathsf{pay}) = \emptyset$

The last three actions are primitive. Note that $\mathsf{Post}(\mathsf{goAB})$ does not mention the possible effect $\neg\mathsf{Money}$, which is only produced when goAB is refined to taxiAB.

An HTN planning domain is captured in PDL by the following theory:

$$\begin{aligned}
\mathtt{Fml}(\mathsf{Pre}) &= \{\langle\alpha\rangle\top \leftrightarrow \mathsf{Pre}(\alpha) \ : \ \alpha \in \mathtt{Act}\} \\
\mathtt{Fml}(\mathsf{Post}) &= \{[\alpha]\mathsf{Post}(\alpha) \ : \ \alpha \in \mathtt{Act}\} \ \cup \{\, p \rightarrow [a]p \ : \ a \in \mathtt{Act}_0 \text{ and } p \notin \mathsf{eff}^{-}(a)\} \\
&\qquad \cup \{\neg p \rightarrow [a]\neg p \ : \ a \in \mathtt{Act}_0 \text{ and } p \notin \mathsf{eff}^{+}(a)\} \\
\mathtt{Fml}(\mathsf{Ref}) &= \{\langle\alpha\rangle\top \rightarrow \pi \sqsubseteq \alpha \ : \ \alpha \in \mathtt{Act}, \pi \in \mathsf{Ref}(\alpha)\}
\end{aligned}$$

So primitive actions behave like STRIPS actions, while high-level actions are less constrained, leaving room for conditional effects and other side effects. The theory of an HTN planning domain is $\mathtt{Fml}(\mathcal{D}_{\mathsf{htn}}) = \mathtt{Fml}(\mathsf{Pre}) \cup \mathtt{Fml}(\mathsf{Post}) \cup \mathtt{Fml}(\mathsf{Ref})$.

3.2 HTN planning problems and their solutions

A *HTN planning problem* is a triple $\mathcal{P}_{\mathsf{htn}} = \langle \mathcal{D}_{\mathsf{htn}}, \mathtt{I}, \pi \rangle$ where $\mathcal{D}_{\mathsf{htn}}$ is an HTN planning domain, $\mathtt{I} \in \mathtt{Fml}_{\mathsf{bool}}$ is a boolean formula, and $\pi \in \mathtt{Pgm}_{\mathsf{PDL}}$ is a program ('initial task network'). For our travelling domain we may e.g. have $\langle \mathcal{D}^{\mathsf{AB}}_{\mathsf{htn}}, \mathtt{I}, \mathsf{goAB} \rangle$ with $\mathtt{I} = \mathsf{AtA} \wedge \neg\mathsf{AtB} \wedge \mathsf{Money}$. (Usually $\mathtt{I}$ is a complete description of a state, but this is not necessary here.)

Traditionally, solutions of $\mathcal{P}_{\mathsf{htn}}$ are obtained by a fixed-point definition, in three steps. First, the *reduction* of a program π is:

$$\mathsf{red}(\mathcal{D}_{\mathsf{htn}}, \pi) = \{\pi^{\alpha}_{\mathsf{Pre}(\alpha)?;\pi'} \ : \ \alpha \text{ occurs in } \pi \text{ and } \pi' \in \mathsf{Ref}(\alpha)\}$$

where $\pi^{\alpha}_{\mathsf{Pre}(\alpha)?;\pi'}$ is obtained from π by replacing some occurrence of α in π by $\mathsf{Pre}(\alpha)?;\pi'$. For the introductory example: $\mathsf{red}(\mathcal{D}_{\mathsf{htn}}, (\beta; p?)) = \{(b; \neg p?; p?)\}$. Second, for a primitive π_0 we define its *completion* as follows:

$$\mathsf{compl}(\mathcal{D}_{\mathsf{htn}}, \mathtt{I}, \pi_0) = \{a_1;\cdots;a_n \ : \ \mathtt{Fml}(\mathsf{Post}) \models \mathtt{I} \rightarrow \langle (a_1;\cdots;a_n) \sqcap \pi_0 \rangle \top\}$$

For example, $\mathsf{compl}(\mathcal{D}_{\mathsf{htn}}, \mathtt{I}, (b; \neg p?; p?)) = \emptyset$. Third, the *solutions* of an HTN planning problem are primitive plans that are defined recursively as follows:

$$\mathsf{sol}^1(\mathcal{D}_{\mathsf{htn}}, \mathtt{I}, \pi) = \begin{cases} \mathsf{compl}(\mathcal{D}_{\mathsf{htn}}, \mathtt{I}, \pi) & \text{if } \pi \text{ is primitive} \\ \emptyset & \text{otherwise} \end{cases}$$

$$\mathsf{sol}^{k+1}(\mathcal{D}_{\mathsf{htn}}, \mathtt{I}, \pi) = \mathsf{sol}^k(\mathcal{D}_{\mathsf{htn}}, \mathtt{I}, \pi) \cup \bigcup_{\pi' \in \mathsf{red}(\mathcal{D}_{\mathsf{htn}}, \pi)} \mathsf{sol}^k(\mathcal{D}_{\mathsf{htn}}, \mathtt{I}, \pi')$$

Letting $\mathsf{sol}(\mathcal{D}_{\mathsf{htn}}, \mathtt{I}, \pi) = \bigcup_k \mathsf{sol}^k(\mathcal{D}_{\mathsf{htn}}, \mathtt{I}, \pi)$ we are able to connect the traditional solutions of HTN planning problems and logical consequence in PDL:

Theorem 1. *If* $a_1;\cdots;a_n \in \mathsf{sol}(\mathcal{D}_{\mathsf{htn}}, \mathtt{I}, \pi)$ *then* $\mathtt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \mathtt{I} \rightarrow \langle (a_1;\cdots;a_n) \sqcap \pi \rangle \top$.

4 Rationality postulates for HTN planning

We now introduce postulates of refinement soundness and completeness. Further postulates of modularity are discussed in the long report.

When α is executable then all refinements of α should guarantee the post-conditions of α. This has to be conditioned: if $\mathsf{Pre}(\alpha)$ is false then there is no point in refining.

Definition 1. *Action* α *is* soundly refinable at (M, w) *if and only if either* $M, w \not\Vdash \mathsf{Pre}(\alpha)$ *or for every* $\pi \in \mathsf{Ref}(\alpha)$ *and* $v \in R_\pi(w)$, $M, v \Vdash \mathsf{Post}(\alpha)$.

Clearly, a reasonable HTN domain should be such that every action is soundly refinable at every pointed model (M, w). This can be characterised in PDL.

Theorem 2. *Let* $\mathcal{D}_{\mathsf{htn}}$ *be an HTN domain. An action* $\alpha \in \mathtt{Act}$ *is soundly refinable at every pointed model* (M, w) *iff* $\mathtt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \mathsf{Pre}(\alpha) \rightarrow \big[\bigsqcup \mathsf{Ref}(\alpha)\big]\mathsf{Post}(\alpha)$.

One may also define complete refinability: when the precondition of a high-level action is true then there should be a way of refining it.

Definition 2. *High-level action* $\alpha \in \mathtt{Act} \setminus \mathtt{Act}_0$ *is* completely refinable at (M, w) *if and only if either* $M, w \not\Vdash \mathsf{Pre}(\alpha)$ *or there is a* $\pi \in \mathsf{Ref}(\alpha)$ *such that* $R_\pi(w) \neq \emptyset$.

In other words, as long as the precondition of α is true, one of the programs refining α should be executable.

Theorem 3. *An action* $\alpha \in \mathtt{Act} \setminus \mathtt{Act}_0$ *is completely refinable at every pointed model* (M, w) *iff* $\mathtt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \mathsf{Pre}(\alpha) \rightarrow \big\langle \bigsqcup \mathsf{Ref}(\alpha)\big\rangle\top$.

As discussed in [12], even when some refinement is physically possible, there may be reasons for not including it in the Ref function. There are two possible such reasons: either the refinement is legally impossible, or it is not preferred. This former case of incompleteness can be illustrated with the help of Example 1: the primitive plan rideAB of taking the taxi without paying also achieves the postconditions of goAB. However, the domain designer did not want to allow such a refinement and deliberately omitted it from Ref(goAB).

Complete refinability can be weakened by requiring refinability *unless there is no primitive plan achieving the postconditions of* α. This is similar to what is called *planner completeness* in [12], which, as we understand it, requires that every solution that can be obtained by a classical planner is also obtainable by the HTN planner. It can be characterized by the PDL formula

$$\mathtt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \big(\mathsf{Pre}(\alpha) \wedge \big\langle \big(\bigsqcup \mathtt{Act}_0\big)^*\big\rangle \mathsf{Post}(\alpha)\big) \rightarrow \big\langle \bigsqcup \mathsf{Ref}(\alpha)\big\rangle\top.$$

5 Conclusion

We have proposed a representation of HTN in PDL with program inclusion, identifying HTN methods with PDL programs. We have formulated soundness and completeness postulates and have characterised them in PDL. It is clear that methods with linear constraints can be expressed in this way by sequential composition and tests. We leave the exact correspondence with more general constraints to future work and just note that the PDL program operators are expressive enough to capture the standard examples in the literature. Given results on grammar logics [2, 4], our extension of PDL is undecidable, and it can be conjectured that fragments corresponding to regular grammars are decidable.

Previous work embedding HTN in the Situation Calculus [1, 6, 7] is discussed in more detail in the long report. Relations between HTN planning with the semantics of BDI logics are investigated in [3, 14, 9, 10].

References

1. Baral, C., Son, T.C.: Extending ConGolog to allow partial ordering. In: Proceedings of the 6th International Workshop on Agent Theories, Architectures, and Languages. pp. 188–204. Springer (1999)
2. Fariñas del Cerro, L., Penttonen, M.: Grammar logics. Logique Et Analyse 31(121-122), 123–134 (1988)
3. De Silva, L., Sardina, S., Padgham, L.: First principles planning in BDI systems. In: Proceedings of the 8th International Conf. on Autonomous Agents and Multiagent Systems (AAMAS). vol. 2, pp. 1105–1112. International Foundation for Autonomous Agents and Multiagent Systems (2009)
4. Demri, S.: The complexity of regularity in grammar logics and related modal logics. J. Log. Comput. 11(6), 933–960 (2001), http://dx.doi.org/10.1093/logcom/11.6.933
5. Erol, K., Hendler, J., Nau, D.S.: HTN planning: Complexity and expressivity. In: Proceedings of the 12th National Conference on Artificial Intelligence (AAAI). vol. 94, pp. 1123–1128 (1994)
6. Gabaldon, A.: Programming hierarchical task networks in the situation calculus. In: Proceedings of the 5th International Conf. on Artificial Intelligence Planning and Scheduling Systems Workshop on On-line Planning and Scheduling (2002)
7. Goldman, R.P.: A semantics for HTN methods. In: Gerevini, A., Howe, A.E., Cesta, A., Refanidis, I. (eds.) Proc. of the 19th International Conference on Automated Planning and Scheduling, (ICAPS). AAAI (2009)
8. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. MIT Press (2000)
9. Herzig, A., Lorini, E., Perrussel, L., Xiao, Z.: BDI logics for BDI architectures: old problems, new perspectives. Künstliche Intelligenz (to appear)
10. Herzig, A., Perrussel, L., Xiao, Z., Zhang, D.: Refinement of intentions. In: Michael, L., Kakas, A.C. (eds.) Proc. JELIA 2016. Springer Verlag (2016)
11. Kambhampati, S., Cutkosky, M.R., Tenenbaum, J.M., Lee, S.H.: Integrating general purpose planners and specialized reasoners: case study of a hybrid planning architecture. IEEE Transactions on Systems, Man, and Cybernetics 23(6), 1503–1518 (1993)
12. Kambhampati, S., Mali, A., Srivastava, B.: Hybrid planning for partially hierarchical domains. In: Proceedings of the 17th National Conference on Artificial Intelligence and 12th Conference on on Innovative Applications of Artificial Intelligence (AAAI/IAAI). pp. 882–888 (1998)
13. Nau, D., Ghallab, M., Traverso, P.: Automated Planning: Theory & Practice. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2004)
14. Sardina, S., de Silva, L., Padgham, L.: Hierarchical planning in BDI agent programming languages: A formal approach. In: Proceedings of the 5th International conf. on Autonomous agents and multiagent systems. pp. 1001–1008. ACM (2006)