M1 CSA
Introduction to Embedded Systems
Model Checking

Jan-Georg Smaus

Université de Toulouse/IRIT

Year 2021/2022

Acknowledgements

The slides and exercises of this chapter are based on material by Marie Duflot-Kremer and Stephan Merz, as well as Stefan Leue.

Plan

Basic Idea of Model Checking

- Analyze the state graph of a given finite system
  - system: algorithm, circuit, protocol, ...
  - represented by a transition system
- Properties to verify:
  - safety: nothing bad will ever happen
  - liveness: something good will eventually happen
- Main application domains:
  - reactive systems: permanent interaction with environment
  - parallel and distributed algorithms, protocols, controllers

Basic Idea of Model Checking

- Analyze the state graph of a given finite system
  - system: algorithm, circuit, protocol, ...
  - represented by a transition system
- Properties to verify:
  - safety: nothing bad will ever happen
  - liveness: something good will eventually happen
- Main application domains:
  - reactive systems: permanent interaction with environment
  - parallel and distributed algorithms, protocols, controllers
- Control is more important than data
- Systems are usually composed of several parts

Three Steps of Model Checking

1. Construct system model
   - describe each system component (e.g., process) by a (finite) automaton
   - languages: TLA^+/ PlusCal, Promela, Petri nets, process algebra, ...
   - possibly: automatic extraction from source code

2. Specification of expected properties by temporal logic. Examples:
   - mutual exclusion $\Box\neg(pc[0] = \text{“cs”} \wedge pc[1] = \text{“cs”})$
   - guaranteed response $(pc[0] = \text{“a2”}) \leadsto (pc[0] = \text{“cs”})$

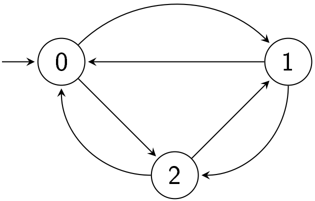
3. Verification
   - “push-button”: automatic verification by model checker
   - failure: examine counter-example to determine why property fails
   - success: property holds for the model
   - memory overflow / timeout: simplify the model

Plan

Plan

General Framework for Modelling Discrete Systems

- Transition system ≈ automaton, without acceptance condition
    - example: counter modulo 3



- Generator of runs
    - run: infinite sequence of states and transitions
    - system properties are evaluated over runs
    - flat model: internal structure of states is not represented
      abstract from variables, processes, communication, ...
    - observe only which state the system is currently in

Transition systems: definition

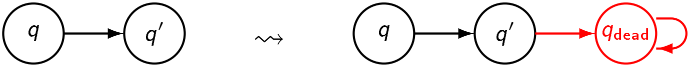
- Abstract model of reactive systems $\mathcal{T} = (Q, I, \delta)$
  - Q — finite set of states
  - $I \subseteq Q$ — initial states
  - $\delta \subseteq Q \times Q$ — (total) transition relation:
    for all $q \in Q$ there exists $q' \in Q$ s.t. $(q, q') \in \delta$

- In practice: $\mathcal{T}$ (i.e., Q and δ) described implicitly
  - TLA^{+}/Promela: state = assignment of values to state variables
  - Petri nets: state = marking of places in the net

- Size of Q is in general exponential in size of the description of $\mathcal{T}$

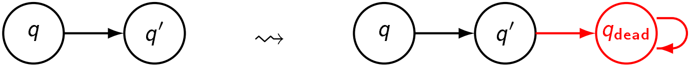
Transition systems: remarks

- Totality of δ
  - technical requirement: simplifies subsequent definitions
  - every finite execution can be extended to an infinite one
  - deadlock must be modelled explicitly



Transition systems: remarks

- Totality of δ
  - technical requirement: simplifies subsequent definitions
  - every finite execution can be extended to an infinite one
  - deadlock must be modelled explicitly

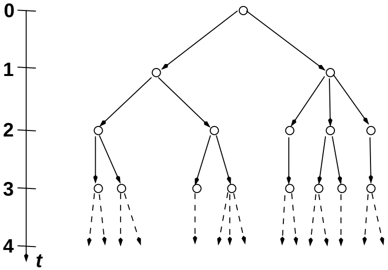


- Variant: labelled transitions $\delta \subseteq Q \times \mathcal{A} \times Q$
  - explicitly identify actions responsible for transitions
  - distinguish internal and communication transitions
  - timed systems, probabilistic systems, . . .   (more later and in M2 V&C)

Runs of Transition Systems

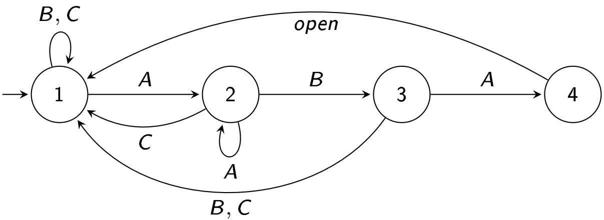
- Run $\rho = q_0 q_1 \ldots$ of $\mathcal{T} = (Q, I, \delta)$
  - $q_0 \in I$ initial state
  - $(q_i, q_{i+1}) \in \delta$ state succession
  - labelled transitions: $\rho = q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} q_2 \ldots$

- Unfolding: tree (or forest) representing all runs of $\mathcal{T}$



nodes	states of $\mathcal{T}$
edges	transitions
paths	runs
branching	non-determinism

Example: Digicode as Labelled Transition System



- Door opens in state 4 and is closed otherwise
- The door opens for any code ending in ABA
- Runs of this transition system
    - sequence of states (and actions) describing system evolution
    - $1 \xrightarrow{B} 1 \xrightarrow{A} 2 \xrightarrow{A} 2 \xrightarrow{C} 1 \ldots$
    - $1 \xrightarrow{A} 2 \xrightarrow{B} 3 \xrightarrow{C} 1 \xrightarrow{C} 1 \xrightarrow{A} 2 \xrightarrow{B} 3 \xrightarrow{A} 4 \xrightarrow{open} 1 \ldots$

Exercise 1

Give another run of this system.

Plan

Augmented Transition Systems

We introduce variables over finite domains.

- system state: automaton state + variable values
- more succinct model, easier to understand

Having variables, it makes sense to annotate transitions:

- guard: predicate over variables restricts transition
- update: change values of some variables upon transition

Augmented Transition Systems

We introduce variables over finite domains.

- system state: automaton state + variable values
- more succinct model, easier to understand

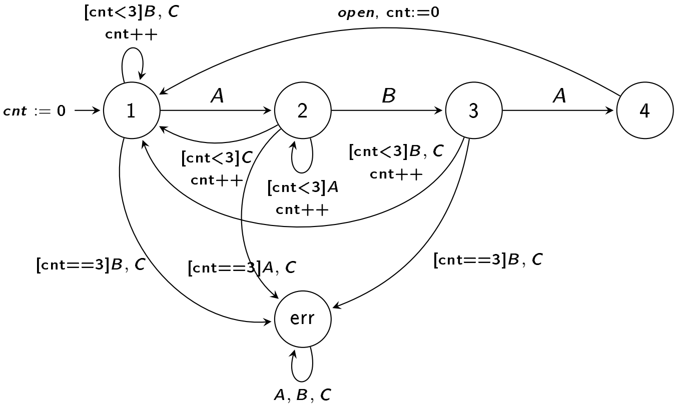
Having variables, it makes sense to annotate transitions:

- guard: predicate over variables restricts transition
- update: change values of some variables upon transition

Same expressiveness as basic transition systems

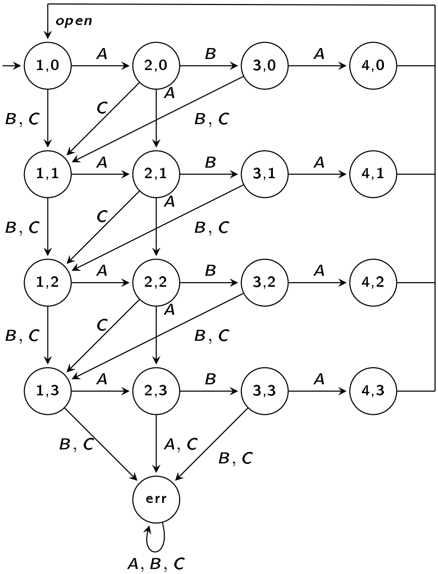
- make as many copies of states as there are values of variables
- evaluate guards and assignments over constant values

Example: Digicode with Counter



- variable *cnt* indicates number of successive erroneous entries
- door remains locked after more than 3 erroneous attempts

Digicode with Counter, Flattened



Exercise 2

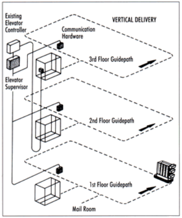
How many states are there, and why?

Plan

Composition of Transition Systems

- Systems are usually built from components
  - parallel programs built from processes
  - hardware built from interacting circuits
  - networked systems built from communicating nodes

Example: an elevator is made of a cabin, doors and a controller.



Composition of Transition Systems (2)

- Assemble overall transition system
  - represent each component by separate transition system
  - derive global transition system from component systems
  - different system paradigms reflected by synchronization schemes; here we consider synchronous composition
- Interest for model checking
  - need not explicitly store global transition system
  - component systems can be much smaller than global system
  - can sometimes benefit from symmetries to reduce state space

Synchronizing (Handshake) Composition

- Synchronization via shared actions
  - assume labelled transition systems $\mathcal{T}_i = (Q_i, \mathcal{A}_i, I_i, \delta_i) \quad (i = 1, 2)$
  - synchronized product $\mathcal{T} = (Q_1 \times Q_2, \mathcal{A}_1 \cup \mathcal{A}_2, I_1 \times I_2, \delta)$

$$((q_1, q_2), a, (q_1', q_2')) \in \delta \quad \text{iff}$$

$$\begin{array}{l} a \in \mathcal{A}_1 \setminus \mathcal{A}_2 \;\text{ and }\; (q_1, a, q_1') \in \delta_1 \;\text{ and }\; q_2' = q_2 \quad \text{or} \\ a \in \mathcal{A}_2 \setminus \mathcal{A}_1 \;\text{ and }\; (q_2, a, q_2') \in \delta_2 \;\text{ and }\; q_1' = q_1 \quad \text{or} \\ a \in \mathcal{A}_1 \cap \mathcal{A}_2 \;\text{ and }\; (q_1, a, q_1') \in \delta_1 \;\text{ and }\; (q_2, a, q_2') \in \delta_2 \end{array}$$

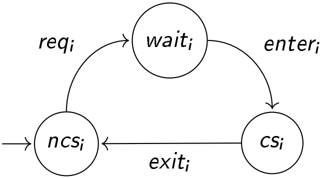
  - joint actions must be executed together, local actions interleave

- Generalizations beyond 2 components
  - multi-party synchronization: actions shared by several components
  - synchronization of two components, the others stutter

Example: Mutual Exclusion by Joint Actions

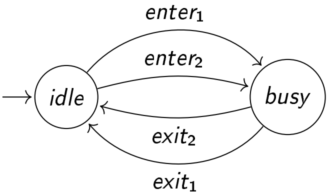
Two processes and a controller

Process P_i $(i = 1, 2)$



$\mathcal{A}_{P_i} = \{req_i, enter_i, exit_i\}$

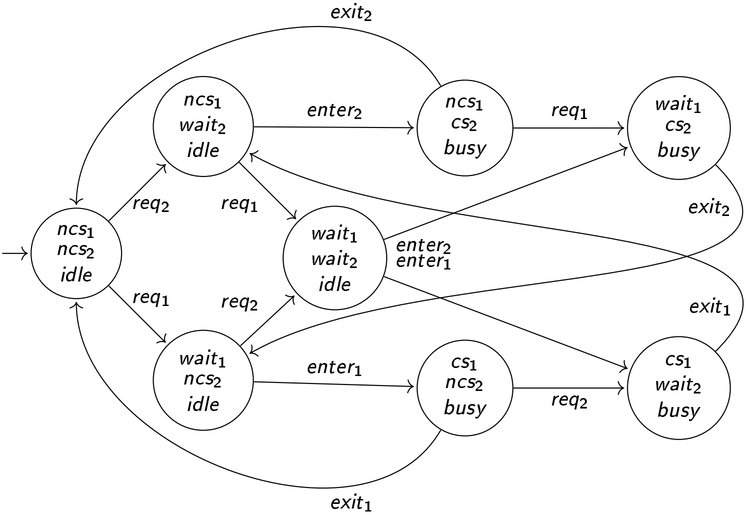
Controller C



$\mathcal{A}_C = \{enter_1, enter_2, exit_1, exit_2\}$

- req_i : local to process P_i
- $enter_i$, $exit_i$: shared between process P_i and controller

Synchronized Product (reachable states)



Exercise 3

How many unreachable states are there?

Plan

From simple to complicated and back

- We have started from plain transition systems: very simple formalism, but system descriptions will be huge and hard to read.

From simple to complicated and back

- We have started from plain transition systems: very simple formalism, but system descriptions will be huge and hard to read.
- We enhanced the formalism to have more concise and readable system descriptions:
  - finite-domain variables with guards and updates;
  - labels (for composition).

We have also seen that in both cases, the enhancements an be “compiled away”, i.e., one can translate an enhanced system into the simple formalism.

From simple to complicated and back

- We have started from plain transition systems: very simple formalism, but system descriptions will be huge and hard to read.
- We enhanced the formalism to have more concise and readable system descriptions:
  - finite-domain variables with guards and updates;
  - labels (for composition).

  We have also seen that in both cases, the enhancements an be "compiled away", i.e., one can translate an enhanced system into the simple formalism.
- Now that we know that this can be done, we want to get back to an abstract, theoretical view and hence the initial simple formalism ...

Plan

Kripke Structures

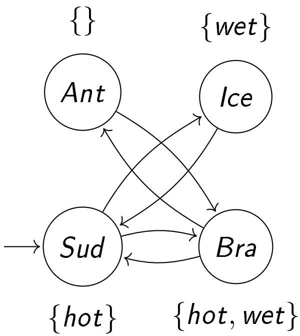
- Add (Boolean) "observations" to the states of a transition system

- Transition system + propositions $\mathcal{K} = (Q, I, \delta, \mathcal{V}, \lambda)$
  - $\mathcal{V}$ set of elementary ("atomic") propositions
  - $\lambda : Q \to 2^{\mathcal{V}}$ $\lambda(q)$ indicates which propositions are true at q

- Atomic propositions
  - "building blocks" for expressing system properties
  - evaluated at states: v is true at q if $v \in \lambda(q)$, false otherwise
  - examples:
    - the door protected by the digicode is open
    - the counter value is at least 3
    - process 0 is at the critical section

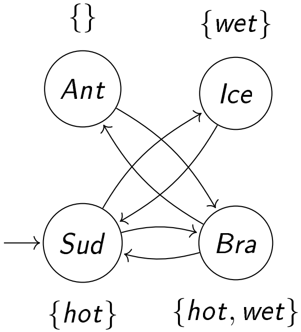
Example of a Kripke Structure

$Q = \{Antarctica, Brazil, Iceland, Sudan\}$.
$I = \{Sudan\}$.
δ as pictured (think of it as "reachability by direct flight").
$\mathcal{V} = \{hot, wet\}$, λ as pictured.



Example of a Kripke Structure

$Q = \{Antarctica, Brazil, Iceland, Sudan\}$.
$I = \{Sudan\}$.
δ as pictured (think of it as "reachability by direct flight").
$\mathcal{V} = \{hot, wet\}$, λ as pictured.



Every run $q_0 q_1 \ldots$ corresponds to an ω-word $\lambda(q_0)\lambda(q_1)\ldots$ over the alphabet $2^{\mathcal{V}}$.
E.g. *Sud Bra Ant Bra Sud Ice* ... corresponds to
$\{hot\}\{hot, wet\}\{\}\{hot, wet\}\{hot\}\{wet\}\ldots$.

Exercise 4

Give another example.

Plan

Plan

Formal Description of Properties

- Need a language for expressing properties of systems
  - system under verification represented as a transition system
  - properties of systems should be expressed unambiguously

- Natural language is ambiguous

Example

"Every student takes a computer science lecture."

- There is a CS lecture taken by all students.
- For every student s there is a CS lecture that s takes.

No obvious interpretation a priori!

- Mathematical logic allows formalizing such statements

Temporal Properties

- Wish to express properties of system executions
  - After the emergency brake is pulled, the train will stop.
  - After subscribing to a phone service, users may receive calls.
  - When the window is broken, an alarm will sound until it is switched off.
  - The lift does not move unless somebody previously requested it.

- Properties on succession of states / events
  - something holds { before / after / between } some other event(s)
  - no references to absolute time (for the moment)

Temporal Properties in First-Order Logic

- Add explicit time parameter
  - propositions can be true or false at different time points t
  - relate different time points (e.g., $t+1$, $t' \geq t$, $\ldots$)

- Example
  - After the emergency brake is pulled, the train will stop.

$$\forall t : \mathit{Brake}(t) \Rightarrow \exists t' : t' \geq t \wedge \mathit{Stop}(t')$$

  - When the window is broken, an alarm will sound until it is switched off.

$$\forall t : \mathit{Break}(t) \Rightarrow \exists t' \geq t : \mathit{Off}(t') \wedge \forall t'' : t \leq t'' < t' \Rightarrow \mathit{Alarm}(t'')$$

- Possible, but somewhat clumsy
  - especially for properties that contain several temporal references
  - moreover, reasoning in first-order logic is undecidable in general

Plan

Linear-Time Temporal Logic: Informally (1)

- Eliminate explicit time parameter
  - formulas are evaluated over infinite state sequences σ
  - they can be true or false at different time points
  - atomic formulas: elementary properties evaluated at states

    $\sigma, i \models \mathit{Break}$ proposition *Break* is true at state i of σ
  - if $\sigma = q_0 q_1 \ldots$ is a run of a Kripke structure $\mathcal{K} = (Q, I, \delta, \mathcal{V}, \lambda)$:

    $\sigma, i \models v$ determined by $\lambda(q_i)$, for $v \in \mathcal{V}$

- Standard Boolean connectives $\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$
  - applied to arbitrary formulas, with standard interpretation
  - $\sigma, j \models \mathit{Alarm} \wedge \neg \mathit{Off}$ *Alarm* true, but *Off* false at state j

Linear-Time Temporal Logic: Informally (2)

- Temporal connectives for temporal references
  - change "point of evaluation" of (sub-)formulas
  - always φ — φ true at all suffixes — $\mathbf{G}\,\varphi$ $(\Box\varphi)$
  - eventually φ — φ true at some suffix — $\mathbf{F}\,\varphi$ $(\Diamond\varphi)$
  - next φ — φ true at immediate suffix — $\mathbf{X}\,\varphi$ $(\circ\varphi)$
  - φ until ψ — φ remains true until ψ becomes true — $\varphi\,\mathbf{U}\,\psi$

- Examples
  - $\mathbf{G}(\mathit{Brake} \Rightarrow \mathbf{F}\ \mathit{Stop})$
  - $\mathbf{G}(\mathit{Break} \Rightarrow (\mathit{Alarm}\ \mathbf{U}\ \mathit{Off}))$

Formal Syntax of LTL

- LTL: compact syntax for properties of runs
  - formulas evaluated over infinite state sequences
  - system satisfies φ if φ holds of every run
- Inductive definition of LTL formulas

$$
\begin{array}{llll}
\varphi & ::= & v \in \mathcal{V} & \text{atomic formulas} \\
& \mid & \neg\varphi, \varphi \vee \varphi & \text{Boolean connectives} \\
& \mid & \mathbf{X}\,\varphi & \text{next state } (\circ\varphi) \\
& \mid & \varphi\,\mathbf{U}\,\varphi & \text{until } (\omega \textbf{ until } \psi)
\end{array}
$$

Exercise 5

How is this notation called?

- Abbreviations
  - $\wedge, \Rightarrow, \Leftrightarrow, \text{true}, \text{false}$ — as in propositional logic
  - $\mathbf{F}\,\varphi \equiv \text{true}\,\mathbf{U}\,\varphi$ — eventually φ (finally, $\Diamond\varphi$)
  - $\mathbf{G}\,\varphi \equiv \neg\,\mathbf{F}\,\neg\varphi$ — always φ (globally, $\Box\varphi$)

Formal Semantics of LTL

- Formulas φ evaluated over infinite sequences of states
  - atomic formulas interpreted by labelling $\lambda : Q \to 2^{\mathcal{V}}$
  - notations: for $\sigma = q_0 q_1 \ldots$, we denote by $\sigma[n..]$ the suffix $q_n q_{n+1} \ldots$.
- Inductive definition of $\sigma \models \varphi$

$$
\begin{array}{lll}
\sigma \models v & \text{iff} & v \in \lambda(\sigma_0) \\
\sigma \models \neg\varphi & \text{iff} & \sigma \not\models \varphi \\
\sigma \models \varphi \vee \psi & \text{iff} & \sigma \models \varphi \ \text{ or } \ \sigma \models \psi \\
\sigma \models \mathbf{X}\,\varphi & \text{iff} & \sigma[1..] \models \varphi \\
\sigma \models \varphi\ \mathbf{U}\ \psi & \text{iff} & \text{there is } k \in \mathbb{N} \text{ such that } \sigma[k..] \models \psi \\
 & & \qquad \text{and } \sigma[i..] \models \varphi \text{ for all } 0 \leq i < k
\end{array}
$$

- Semantics of derived temporal connectives
  - $\sigma \models \mathbf{F}\,\varphi$ iff $\sigma[k..] \models \varphi$ for some $k \in \mathbb{N}$
  - $\sigma \models \mathbf{G}\,\varphi$ iff $\sigma[k..] \models \varphi$ for all $k \in \mathbb{N}$

Example: Interpretation of LTL Formulas

Exercise 6

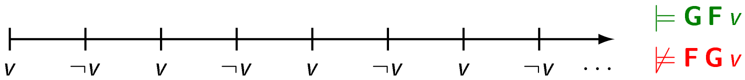
Which of the following formulas are true? (true = 1, false = 0)

hot	0	0	1	1	0	1	1	0	1	$\cdots$	(always 1)
wet	1	1	0	0	0	0	1	1	0	$\cdots$	(always 0)

- $\mathbf{G}(\neg hot \Rightarrow wet)$
- $\mathbf{F}(hot \wedge \neg wet)$
- $\neg hot \ \mathbf{U} \ \neg wet$
- $\mathbf{G}(\neg hot \Rightarrow (\neg hot \ \mathbf{U} \ \neg wet))$
- $\mathbf{G}\,\mathbf{F}(wet)$
- $\mathbf{F}\,\mathbf{G}(\neg hot \Rightarrow wet)$

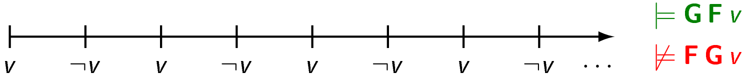
Infinitely Often and Persistence

- $\mathbf{G}\,\mathbf{F}\,\varphi$
  - for every suffix there is a subsequent suffix satisfying φ
  - φ is infinitely often true
- $\mathbf{F}\,\mathbf{G}\,\varphi$
  - there is a suffix such that all subsequent suffixes satisfy φ
  - φ is false only finitely often, φ is persistent
- $\mathbf{F}\,\mathbf{G}\,\varphi$ is strictly stronger than $\mathbf{G}\,\mathbf{F}\,\varphi$



Infinitely Often and Persistence

- $\mathbf{G}\,\mathbf{F}\,\varphi$
  - for every suffix there is a subsequent suffix satisfying φ
  - φ is infinitely often true
- $\mathbf{F}\,\mathbf{G}\,\varphi$
  - there is a suffix such that all subsequent suffixes satisfy φ
  - φ is false only finitely often, φ is persistent
- $\mathbf{F}\,\mathbf{G}\,\varphi$ is strictly stronger than $\mathbf{G}\,\mathbf{F}\,\varphi$



- Combinations
  - $\mathbf{G}(req \Rightarrow \mathbf{F}\,get)$ every request will be satisfied
  - $\mathbf{G}(\mathbf{G}\,req \Rightarrow \mathbf{F}\,get)$ every persistent request will be satisfied
  - $\mathbf{G}(\mathbf{G}\,\mathbf{F}\,req \Rightarrow \mathbf{F}\,get)$ every repeated request will be satisfied

Exercise: Properties of Binary Consensus

Consider a system of N processes $P_1, \ldots, P_N$ where the state of P_i is given by a Boolean variable v_i, indicating its *local value*, and d_i, indicating if P_i has *decided* (initially false). The *Consensus problem* consists in arriving at a state where every process has decided and where the local values of all processes are identical. This is expressed by the four following properties.

1. Validity. At any state, any value v_i must equal the initial value of some v_j (i.e., no values other than those initially present are introduced).
2. Irrevocability. Once P_i decides (i.e., sets its variable d_i to true), the variables v_i and d_i never change again.
3. Agreement. Any two processes P_i and P_j that have decided agree on the values of v_i and v_j.
4. Termination. Every process P_i decides eventually.

Exercise 7

Express these four properties by LTL formulas.

Typical Properties in LTL

- invariants $\mathbf{G}\, p$

  $\mathbf{G}\, \neg(crit_1 \wedge crit_2)$ mutual exclusion

  $\mathbf{G}(pre_1 \vee \ldots \vee pre_n)$ deadlock freedom

- reply, recurrence $\mathbf{G}(p \Rightarrow \mathbf{F}\, q)$

  $\mathbf{G}(try_1 \Rightarrow \mathbf{F}\, crit_1)$ guaranteed access to critical section

  $\mathbf{G}(\quad \mathbf{F}\, \neg crit_1)$ finite stay in critical section

- reactivity $\mathbf{G}\,\mathbf{F}\, p \Rightarrow \mathbf{G}\,\mathbf{F}\, q$

  $\mathbf{G}\,\mathbf{F}(try_1 \wedge \neg crit_2) \Rightarrow \mathbf{G}\,\mathbf{F}\, crit_1$ (strong) fairness

- precedence $p_1\ \mathbf{U} \ldots \mathbf{U}\ p_n$

  $\mathbf{G}(try_1 \wedge try_2 \Rightarrow \neg crit_2\ \mathbf{U}\ crit_2\ \mathbf{U}\ \neg crit_2\ \mathbf{U}\ crit_1)$ 1-bounded overtaking

Fairness Conditions

- Interleaving of parallel processes modelled by non-determinism
  - example: choice between execution of two processes

    algorithm Stopwatch {
        variables $x = 0,\ y = 0$;
        process (w = "watch") {
          α : **while** $(y = 0)$ { $\beta : x := x + 1$; }
        }
        process (s = "stop") {
          $\gamma : \ y := 1$
        }
    }

  - the transition system has a run where γ never happens
  - Arguably, such a run may be considered "unrealistic" and be excluded "by assumption".

Fairness Conditions

- Interleaving of parallel processes modelled by non-determinism
  - example: choice between execution of two processes

```
algorithm Stopwatch {
    variables x = 0, y = 0;
    process (w = "watch") {                          process (s = "stop") {
     α :  while (y = 0) {  β : x := x + 1;  }          γ :  y := 1
    }                                                }
}
```

  - the transition system has a run where γ never happens
  - Arguably, such a run may be considered "unrealistic" and be excluded "by assumption".

- Fairness hypothesis exclude "unfair" runs
  - if an action is possible often enough, it will eventually happen
  - restriction on infinite runs, not on local choice
  - different interpretations of "often enough"

Fairness in LTL

- Weak fairness
  - if the action is always enabled, it will eventually happen
  - $\mathrm{WF}(A) \equiv \mathbf{G}(\mathbf{G}\, enabled_A \Rightarrow \mathbf{F}\, taken_A)$

Fairness in LTL

- Weak fairness
  - if the action is always enabled, it will eventually happen
  - $\mathrm{WF}(A) \equiv \mathbf{G}(\mathbf{G}\, enabled_A \Rightarrow \mathbf{F}\, taken_A)$
- Strong fairness
  - if the action is enabled infinitely often, it will eventually happen
  - $\mathrm{SF}(A) \equiv \mathbf{G}(\mathbf{G}\,\mathbf{F}\, enabled_A \Rightarrow \mathbf{F}\, taken_A)$

Fairness in LTL

- Weak fairness
  - if the action is always enabled, it will eventually happen
  - $\mathrm{WF}(A) \equiv \mathbf{G}(\mathbf{G}\, enabled_A \Rightarrow \mathbf{F}\, taken_A)$
- Strong fairness
  - if the action is enabled infinitely often, it will eventually happen
  - $\mathrm{SF}(A) \equiv \mathbf{G}(\mathbf{G}\,\mathbf{F}\, enabled_A \Rightarrow \mathbf{F}\, taken_A)$
- System verification under fairness hypotheses
  - include hypothesis in the formula expressing the property
  - example: $\mathrm{WF}(Exit_2) \wedge \mathrm{SF}(Enter_1) \Rightarrow \mathbf{G}(try_1 \Rightarrow \mathbf{F}\, crit_1)$

It is like saying: a coin tossed infinitely often will eventually show “heads”.

Exercise 8

How would you program a coin tossing simulator giving an infinite sequence such that

- for any n, the first n tosses will be “tails”, with probability > 0;
- Eventually a toss will be “heads”?

Plan

Plan

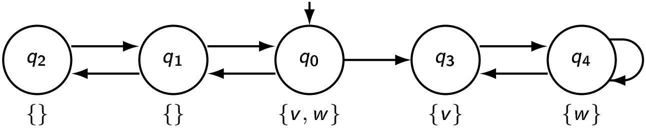
$\mathcal{K}$-Validity

Given a Kripke structure $\mathcal{K}$, we write $\mathcal{K} \models \varphi$ iff for every run $q_0 q_1 \ldots$ of $\mathcal{K}$, we have $q_0 q_1 \ldots \models \varphi$.

$\mathcal{K}$-Validity

Given a Kripke structure $\mathcal{K}$, we write $\mathcal{K} \models \varphi$ iff for every run $q_0 q_1 \ldots$ of $\mathcal{K}$, we have $q_0 q_1 \ldots \models \varphi$.

Exercise: Consider the following Kripke structure $\mathcal{K}$.



Exercise 9

Determine if $\mathcal{K} \models \varphi_i$ holds for the following formulas φ_i. Briefly justify your answers.

$$
\begin{array}{lcl@{\qquad}lcl}
\varphi_1 & = & \mathbf{G}(v \Rightarrow w \vee \mathbf{X}\, w) & \varphi_6 & = & \mathbf{G}\,\mathbf{F}\, w \\
\varphi_2 & = & \neg w \Rightarrow (\neg w \;\mathbf{U}\; w) & \varphi_7 & = & \mathbf{G}\,\mathbf{F}\, \neg v \\
\varphi_3 & = & \mathbf{G}(\neg w \Rightarrow (\neg w \;\mathbf{U}\; w)) & \varphi_8 & = & \mathbf{G}\,\mathbf{F}\, \neg w \\
\varphi_4 & = & \mathbf{G}\,\mathbf{F}\, v & \varphi_9 & = & (\mathbf{G}\,\mathbf{F}\, v) \Rightarrow (\mathbf{G}\,\mathbf{F}\, w) \\
\varphi_5 & = & \mathbf{G}\,\mathbf{F}(v \wedge w) & \varphi_{10} & = & (\mathbf{G}\,\mathbf{F}(v \wedge w)) \Rightarrow (\mathbf{G}\,\mathbf{F}(v \wedge \neg w))
\end{array}
$$

Example: Verifying a Persistence Property

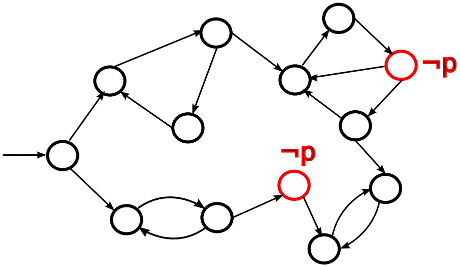
Consider the problem of verifying $\mathcal{K} \models \mathbf{F}\,\mathbf{G}\,v$.

- The property is violated iff there exists a run $\sigma = q_0 q_1 \dots q_{i_0} \dots q_{i_1} \dots q_{i_2} \dots$ such that $v \notin \lambda(q_{i_j})$ for all $j \in \mathbb{N}$. Since $\mathcal{K}$ is finite-state, we must have $q_{i_j} = q_{i_k}$ for some $k > j$.
- The prefix of σ given by $q_0 \Rightarrow^* q_{i_j} \Rightarrow^+ q_{i_k}$ is effectively a "lasso" through a state where v is false

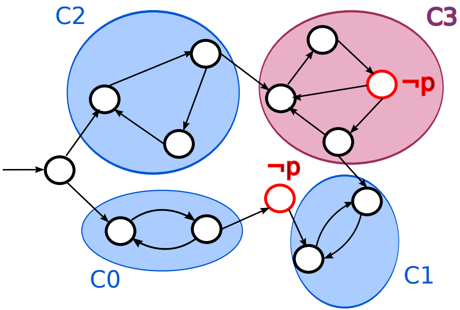
To search for a lasso, we inspect graph G of reachable states of $\mathcal{K}$:

- compute strongly connected components of G (can be done by Tarjan's algorithm: linear in size of G).
- for each component, check if it contains some q with $v \notin \lambda(q)$. If you find such a component, you have found a lasso. Hence $\mathcal{K} \not\models \mathbf{F}\,\mathbf{G}\,v$.

Example: Verifying **F G** p



Example: Verifying **F G** p

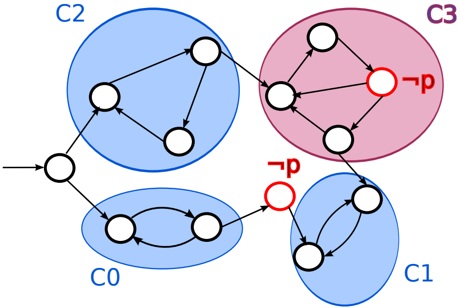


$\mathcal{K} \not\models \mathbf{F}\,\mathbf{G}\,p$: component C3 contains state where p is false.

Exercise 10

Why is the occurrence of $\neg p$ in C3 decisive, why does the other occurrence not matter?

Example: Verifying **F G** p



$\mathcal{K} \not\models \mathbf{F}\,\mathbf{G}\,p$: component C3 contains state where p is false.

Exercise 10

Why is the occurrence of $\neg p$ in C3 decisive, why does the other occurrence not matter?

But it is not obvious how to generalize this idea to arbitrary LTL properties.

Principle of LTL Model Checking

- Make use of automata theory
  - view sequence σ of states as ω-word over alphabet $2^{\mathcal{V}}$
  - view LTL formula φ as describing a language $\mathcal{L}(\varphi)$
  - construct automaton $\mathcal{A}_\varphi$ with $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$
  - view $\mathcal{K}$ as generating a language $\mathcal{L}(\mathcal{K})$

Principle of LTL Model Checking

- Make use of automata theory
  - view sequence σ of states as ω-word over alphabet $2^{\mathcal{V}}$
  - view LTL formula φ as describing a language $\mathcal{L}(\varphi)$
  - construct automaton $\mathcal{A}_\varphi$ with $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$
  - view $\mathcal{K}$ as generating a language $\mathcal{L}(\mathcal{K})$

$$\boxed{\begin{array}{c} \mathcal{K} \models \varphi \\ \text{iff} \\ \mathcal{L}(\mathcal{K}) \subseteq \mathcal{L}(\varphi) \\ \text{iff} \\ \mathcal{L}(\mathcal{K}) \cap \mathcal{L}(\neg\varphi) = \emptyset \\ \text{iff} \\ \mathcal{L}(\mathcal{K} \times \mathcal{A}_{\neg\varphi}) = \emptyset \end{array}}$$

Principle of LTL Model Checking

- Make use of automata theory
  - view sequence σ of states as ω-word over alphabet $2^{\mathcal{V}}$
  - view LTL formula φ as describing a language $\mathcal{L}(\varphi)$
  - construct automaton $\mathcal{A}_\varphi$ with $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$
  - view $\mathcal{K}$ as generating a language $\mathcal{L}(\mathcal{K})$

$$\boxed{\begin{array}{c} \mathcal{K} \models \varphi \\ \text{iff} \\ \mathcal{L}(\mathcal{K}) \subseteq \mathcal{L}(\varphi) \\ \text{iff} \\ \mathcal{L}(\mathcal{K}) \cap \mathcal{L}(\neg\varphi) = \emptyset \\ \text{iff} \\ \mathcal{L}(\mathcal{K} \times \mathcal{A}_{\neg\varphi}) = \emptyset \end{array}}$$

- Must solve two main problems (for an appropriate class of automata)
  - translation of formulas $\psi \rightsquigarrow \mathcal{A}_\psi$ (see next ...)
  - decide emptiness problem $\mathcal{L}(\mathcal{A}) \stackrel{?}{=} \emptyset$ (see Sec. 5.3)

Plan

Büchi Automata (General Definition)

- Finite automata operating over ω-words over any alphabet Σ
  $\mathcal{B} = (S, I, \delta, F)$

$$\left.\begin{array}{ll} S & \text{finite set of states} \\ I \subseteq S & \text{initial states} \\ \delta \subseteq S \times \Sigma \times S & \text{transition relation} \\ F \subseteq S & \text{accepting states} \end{array}\right\} \text{just like ordinary nondeterministic finite automaton}$$

- Run $\rho = s_0 s_1 s_2 \ldots$ of $\mathcal{B}$ over word $\sigma_0 \sigma_1 \sigma_2 \ldots \in \Sigma^\omega$
  - initialization $s_0 \in I$
  - succession $(s_i, \sigma_i, s_{i+1}) \in \delta$ for all $i \in \mathbb{N}$
  - acceptance $s_i \in F$ for infinitely many $i \in \mathbb{N}$

- Languages
  - $\mathcal{L}(\mathcal{B})$: language of automaton $\mathcal{B}$ = set of words for which there exists an accepting run
  - ω-regular languages = languages definable by Büchi automata

Büchi Automata for Model Checking

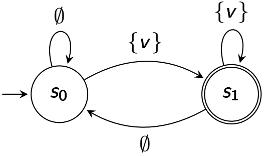
For model checking, we work with an unusual alphabet: $\Sigma = 2^{\mathcal{V}}$.

Run $\rho = s_0 s_1 s_2 \ldots$ of $\mathcal{B}$ over word $L_0 L_1 L_2 \ldots \in (2^{\mathcal{V}})^{\omega}$

- initialization $s_0 \in I$
- succession $(s_i, L_i, s_{i+1}) \in \delta$ for all $i \in \mathbb{N}$
- acceptance $s_i \in F$ for infinitely many $i \in \mathbb{N}$

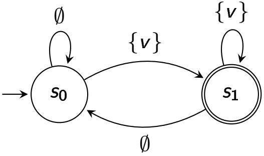
Displaying Büchi Automata

Let $\mathcal{V} = \{v\}$. For example, consider the automaton

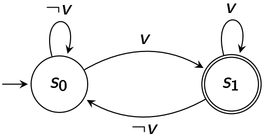


Displaying Büchi Automata

Let $\mathcal{V} = \{v\}$. For example, consider the automaton



It may also be displayed conveniently using some logical notation:

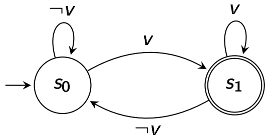


We will use such notation in the sequel.

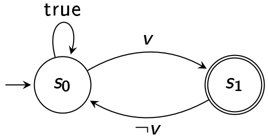
Exercise 11

Explain in a couple of words the language of this automaton.

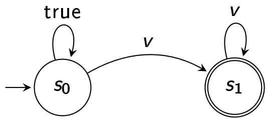
Examples of Büchi Automata
(with Semi-formal Language Description)

- 

infinitely often v

- 

infinitely often "$v \wedge \mathbf{X} \neg v$"

- 

eventually always v (cannot be expressed by deterministic Büchi automaton)

Exercise 12

Why not?

Semi-formal = resembling LTL

Exercise

Exercise 13

Define Büchi automata that accept precisely the structures satisfying the following LTL formulas. A graphical representation of the automata is sufficient.

You may define the automata in an ad-hoc way.

- $\mathbf{F}\,\mathbf{G}\,v \wedge \mathbf{F}\,\mathbf{G}\,w$
- $\mathbf{G}\,\mathbf{F}\,v \wedge \mathbf{G}\,\mathbf{F}\,\neg v$

Büchi Automata vs. Kripke Structures

Kripke structures and Büchi automata are similar concepts. Both have runs one can associate with ω-words on $2^{\mathcal{V}}$. But there is one difference:

- In Kripke structures, each state is labelled with a property set;
- in Büchi automata, each transition is labelled with a property set.

This is a technical complication making it non-obvious to define $\mathcal{K} \times \mathcal{A}_{\neg\varphi}$, but it is doable. We do not give details here.

Plan

From LTL to Büchi automata

- We have seen on some simple examples how a Büchi automaton for an LTL formula is constructed.
  Now we look at the general construction.

- Idea of construction of generalized (see later) Büchi automaton:
  - automaton states: sets of sub-formulas "promised to be true"
  - decompose every formula in one part to be satisfied now and another part to be satisfied from successor state onwards
  - accepting states determined by subformulas φ **U** ψ

From LTL to Büchi automata

- We have seen on some simple examples how a Büchi automaton for an LTL formula is constructed.
  Now we look at the general construction.

- Idea of construction of generalized (see later) Büchi automaton:
  - automaton states: sets of sub-formulas "promised to be true"
  - decompose every formula in one part to be satisfied now and another part to be satisfied from successor state onwards
  - accepting states determined by subformulas $\varphi \; \mathbf{U} \; \psi$

- Size of automaton: $2^{O(|\varphi|)}$ ($|\varphi|$: length of φ)
  - in the following: suboptimal construction that is easy to define

Automaton States

- Suppose that φ is in positive form
  - negation only applied to atomic formulas
  - transformation possible using dual operators

$$\neg(\varphi \wedge \psi) \Leftrightarrow \neg\varphi \vee \neg\psi \qquad \neg\,\mathbf{G}\,\varphi \Leftrightarrow \mathbf{F}\,\neg\varphi \qquad \text{etc.}$$

Automaton States

- Suppose that φ is in positive form
  - negation only applied to atomic formulas
  - transformation possible using dual operators

    $\neg(\varphi \wedge \psi) \Leftrightarrow \neg\varphi \vee \neg\psi \qquad \neg\,\mathbf{G}\,\varphi \Leftrightarrow \mathbf{F}\,\neg\varphi \qquad$ etc.

- States of automaton $\mathcal{A}_\varphi$
  - A state is identified by a set of subformulas of φ, i.e., every $s \subseteq sf(\varphi)$ is (potentially) a state
  - coherent promise w.r.t. current state
    - false $\notin s$
    - not ($v \in s$ and $\neg v \in s$) for $v \in \mathcal{V}$
    - $(\psi_1 \wedge \psi_2) \in s$ iff $\psi_1 \in s$ and $\psi_2 \in s$ for $\psi_1 \wedge \psi_2 \in sf(\varphi)$
    - $(\psi_1 \vee \psi_2) \in s$ iff $\psi_1 \in s$ or $\psi_2 \in s$ for $\psi_1 \vee \psi_2 \in sf(\varphi)$
    - $(\psi_1\ \mathbf{U}\ \psi_2) \in s$ implies $\psi_1 \in s$ or $\psi_2 \in s$
    - $\mathbf{G}\,\psi \in s$ implies $\psi \in s$
  - initial states: states containing φ

Transitions of $\mathcal{A}_\varphi$

- Verify satisfaction of atomic formulas
- Labels of successor states compatible with "recursion laws"

$$\mathbf{G}\,\varphi \;\Leftrightarrow\; \varphi \wedge \mathbf{X}\,\mathbf{G}\,\varphi, \quad \mathbf{F}\,\varphi \;\Leftrightarrow\; \varphi \vee \mathbf{X}\,\mathbf{F}\,\varphi, \quad \varphi\,\mathbf{U}\,\psi \;\Leftrightarrow\; \psi \vee (\varphi \wedge \mathbf{X}(\varphi\,\mathbf{U}\,\psi))$$

- Formal definition $(s, L, s') \in \delta$ iff
  - $L = s \cap \mathcal{V}$ (L satisfies promise of s w.r.t. atomic formulas)
  - $\mathbf{X}\,\psi \in s$ implies $\psi \in s'$
  - $\mathbf{G}\,\psi \in s$ implies $\mathbf{G}\,\psi \in s'$
  - $\mathbf{F}\,\psi \in s$ and $\psi \notin s$ implies $\mathbf{F}\,\psi \in s'$
  - $\psi_1\,\mathbf{U}\,\psi_2 \in s$ and $\psi_2 \notin s$ implies $\psi_1\,\mathbf{U}\,\psi_2 \in s'$

Exercise 14

Explain each of these points in some words.

Example: Automaton for $\mathbf{F}\,\mathbf{G}\,v$

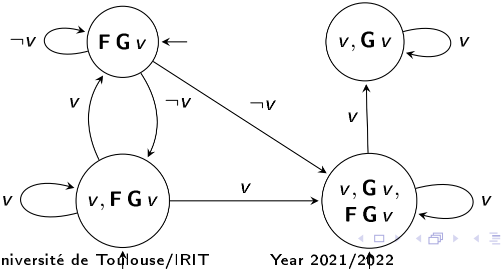
- Subformulas $\quad sf(\mathbf{F}\,\mathbf{G}\,v) = \{v, \mathbf{G}\,v, \mathbf{F}\,\mathbf{G}\,v\}$
- Coherence condition $\quad \mathbf{G}\,v \in s$ implies $v \in s$
  - states respecting coherence conditions

    $\emptyset, \{v\}, \{\mathbf{F}\,\mathbf{G}\,v\}, \{v, \mathbf{F}\,\mathbf{G}\,v\}, \{v, \mathbf{G}\,v\}, \{v, \mathbf{G}\,v, \mathbf{F}\,\mathbf{G}\,v\}$

Exercise 15

What are the potential states that are excluded for incoherence, and why?

  - initial states: $\{\mathbf{F}\,\mathbf{G}\,v\}, \{v, \mathbf{F}\,\mathbf{G}\,v\}, \{v, \mathbf{G}\,v, \mathbf{F}\,\mathbf{G}\,v\}$
- Resulting automaton (reachable part)



Footnote: Generalized Büchi Automata

- Multiple acceptance sets: $\mathcal{B} = (S, I, \delta, \mathcal{F})$
  - S, I, δ: as before
  - $\mathcal{F} = \{F_1, \ldots, F_m\}$: several sets of accepting states
  - run accepting if it visits infinitely often every F_i

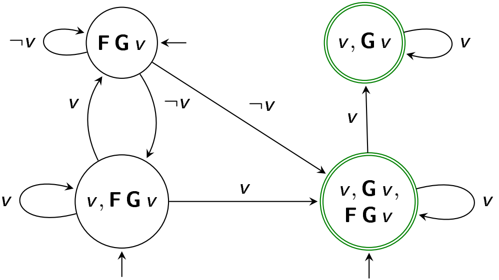
- Encoding into ordinary Büchi automaton $\mathcal{B}' = (S', I', \delta', F')$: see Section 5.2.

Acceptance Conditions

- Objective: exclude loops that do not keep temporal promises
  - relevant subformulas: $\varphi\ \mathbf{U}\ \psi$ (in particular, $\mathbf{F}\ \psi$)
  - must ensure that ψ will eventually be satisfied
- Generalized Büchi condition
  - one acceptance set per subformula $\varphi\ \mathbf{U}\ \psi$
  - $F_{\varphi \mathbf{U} \psi} = \{s : (\varphi\ \mathbf{U}\ \psi) \notin s \text{ or } \psi \in s\}$
  - in particular: $F_{\mathbf{F}\ \psi} = \{s : \mathbf{F}\ \psi \notin s \text{ or } \psi \in s\}$

Acceptance Conditions

- Objective: exclude loops that do not keep temporal promises
  - relevant subformulas: $\varphi\ \mathbf{U}\ \psi$ (in particular, $\mathbf{F}\ \psi$)
  - must ensure that ψ will eventually be satisfied
- Generalized Büchi condition
  - one acceptance set per subformula $\varphi\ \mathbf{U}\ \psi$
  - $F_{\varphi \mathbf{U} \psi} = \{s : (\varphi\ \mathbf{U}\ \psi) \notin s \text{ or } \psi \in s\}$
  - in particular: $F_{\mathbf{F}\,\psi} = \{s : \mathbf{F}\ \psi \notin s \text{ or } \psi \in s\}$
- Example automaton: one acceptance set for $\mathbf{F}\ \mathbf{G}\ v$



Plan

LTL Model Checking Summarised

- Decide $\mathcal{K} \overset{?}{\models} \varphi$
  - view $\mathcal{K}$ as a "Büchi automaton" with trivial acceptance condition
  - $\mathcal{K} \models \varphi$ iff $\mathcal{L}(\mathcal{K}) \cap \mathcal{L}(\mathcal{B}_{\neg\varphi}) = \emptyset$ iff $\mathcal{L}(\mathcal{K} \times \mathcal{B}_{\neg\varphi}) = \emptyset$
  - $\sigma \in \mathcal{L}(\mathcal{K} \times \mathcal{B}_{\neg\varphi})$: counter-example
  - complexity: $O(|\mathcal{K}| \cdot |\mathcal{B}_{\neg\varphi}|)$ (linear in $|\mathcal{K}|$, exponential in $|\varphi|$)

- In practice
  - $|\mathcal{K}|$ is often the critical factor: $|\varphi|$ is often small
  - $\mathcal{K} \times \mathcal{B}_{\neg\varphi}$ can be constructed "on the fly"
  - avoid full computation of product (and its storage in memory)

Complexity is a big issue . . .

Plan

Plan

5. Appendix: More Details on Model Checking
   - Closure Properties of ω-Regular Languages
   - Translation of Generalized Büchi Automata
   - Deciding Emptiness
   - Optimisations

Set-Theoretic Closure of ω-Regular Languages

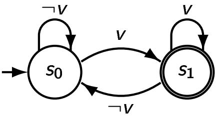
As for the theory of regular languages (finite automata), one has an important property: ω-regular languages are closed under set-theoretic operations $\cup$, $\cap$, complement.
But these are difficult results.

Set-Theoretic Closure of ω-Regular Languages

1. Given Büchi automata $\mathcal{B}_i = (S_i, I_i, \delta_i, F_i)$ $(i = 1, 2,$ where $S_1 \cap S_2 = \emptyset)$, construct a Büchi automaton that accepts the language $\mathcal{L}(\mathcal{B}_1) \cup \mathcal{L}(\mathcal{B}_2)$.
2. Consider Büchi automata $\mathcal{B}_1$ that accepts if v is true infinitely often and $\mathcal{B}_2$ that accepts if v is false infinitely often. Show that the standard product construction does not correspond to language intersection.

   Modify the product construction appropriately. (Hint: introduce a flag that indicates which automaton should accept next.)
3. For a Büchi automaton $\mathcal{B} = (Q, I, \delta, F)$ define $\tilde{\mathcal{B}} = (Q, I, \delta, Q \setminus F)$.

   Construct the automaton $\tilde{\mathcal{B}}$ for the automaton

   Conclude that $\tilde{\mathcal{B}}$ does not define the complement language of a deterministic Büchi automaton $\mathcal{B}$.

   The complement of an ω-regular language is ω-regular: difficult result [Büchi 1960, Safra 1988, Kupferman-Vardi 2001].

Plan

5. Appendix: More Details on Model Checking
   - Closure Properties of ω-Regular Languages
   - Translation of Generalized Büchi Automata
   - Deciding Emptiness
   - Optimisations

Generalized Büchi Automata

- Multiple acceptance sets: $\mathcal{B} = (S, I, \delta, \mathcal{F})$
  - S, I, δ: as before
  - $\mathcal{F} = \{F_1, \dots, F_m\}$: several sets of accepting states
  - run accepting if it visits infinitely often every F_i

- Encoding into ordinary Büchi automaton $\mathcal{B}' = (S', I', \delta', F')$
  - use "counter" indicating which F_i to visit next:

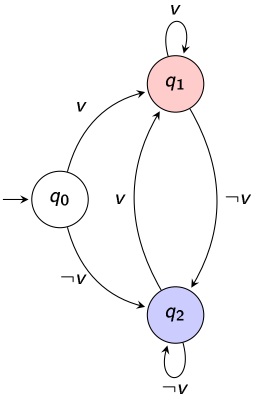
    $S' = S \times \{1, \dots m\}, \ I' = I \times \{1\}$
  - increment counter when designated acceptance set is visited

    $$((s, k), L, (s', k')) \in \delta' \quad \Leftrightarrow \quad \begin{array}{l} (s, L, s') \in \delta \ \text{ and} \\ k' = k \text{ if } s \notin F_k, \\ k' = (k \bmod m) + 1 \text{ otherwise} \end{array}$$
  - acceptance states: states in F_1 with counter value 1

    $F' = F_1 \times \{1\}$

Example: Generalized Büchi Automaton

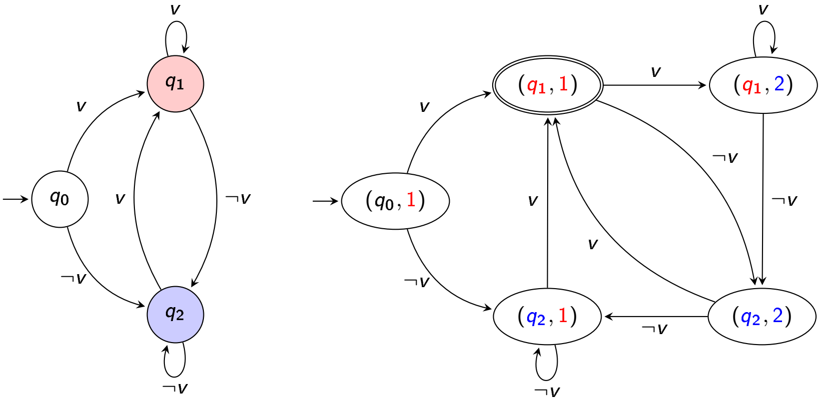
- Recognizing structures satisfying $\mathbf{G}\,\mathbf{F}\,v \wedge \mathbf{G}\,\mathbf{F}\,\neg v$



GBA with $\mathcal{F} = \{\{q_1\}, \{q_2\}\}$

Example: Generalized Büchi Automaton

- Recognizing structures satisfying $\mathbf{G}\,\mathbf{F}\, v \wedge \mathbf{G}\,\mathbf{F}\, \neg v$



GBA with $\mathcal{F} = \{\{q_1\}, \{q_2\}\}$ corresponding ordinary Büchi automaton

Plan

5. Appendix: More Details on Model Checking
   - Closure Properties of ω-Regular Languages
   - Translation of Generalized Büchi Automata
   - Deciding Emptiness
   - Optimisations

Deciding Emptiness

Theorem

For $\mathcal{B} = (S, I, \delta, F)$, its language $\mathcal{L}(\mathcal{B})$ is non-empty iff there are $s \in I$, $s' \in F$ such that $s \Rightarrow^* s'$ and $s' \Rightarrow^+ s'$.

Proof (idea).

$\Leftarrow$: easy

$\Rightarrow$: Assume $\sigma = L_0 L_1 \ldots \in \mathcal{L}(\mathcal{B})$, by accepting run $\rho = s_0 s_1 \ldots$ of $\mathcal{B}$ over σ.

Obviously: $s_0 \in I$

Moreover: some $s \in F$ appears infinitely often in ρ.

Let $k < l$ with $s_k = s_l \in F$: we have $s_0 \Rightarrow^* s_k$ and $s_k \Rightarrow^+ s_k$. q.e.d.

Deciding Emptiness

Theorem

For $\mathcal{B} = (S, I, \delta, F)$, its language $\mathcal{L}(\mathcal{B})$ is non-empty iff there are $s \in I$, $s' \in F$ such that $s \Rightarrow^* s'$ and $s' \Rightarrow^+ s'$.

Proof (idea).

$\Leftarrow$: easy

$\Rightarrow$: Assume $\sigma = L_0 L_1 \ldots \in \mathcal{L}(\mathcal{B})$, by accepting run $\rho = s_0 s_1 \ldots$ of $\mathcal{B}$ over σ.

Obviously: $s_0 \in I$

Moreover: some $s \in F$ appears infinitely often in ρ.

Let $k < l$ with $s_k = s_l \in F$: we have $s_0 \Rightarrow^* s_k$ and $s_k \Rightarrow^+ s_k$. q.e.d.

Implementation:

- enumerate strongly connected components of automaton graph
- determine if some component contains an accepting state
- complexity linear in the size of $\mathcal{B}$: Tarjan's algorithm

Plan

5 Appendix: More Details on Model Checking
- Closure Properties of ω-Regular Languages
- Translation of Generalized Büchi Automata
- Deciding Emptiness
- Optimisations

On-The-Fly Model Checking Algorithm

- Construct the reachable part of $\mathcal{K} \times \mathcal{B}_{\neg\varphi}$
- Construct pairs (q, s) of states of $\mathcal{K}$ and of $\mathcal{B}_{\neg\varphi}$
  - initialization — initial states for both components
  - succession — respect both transition relations $(q, q') \in \delta_{\mathcal{K}}$ and $(s, \lambda(q), s') \in \delta_{\mathcal{B}}$
  - acceptance — pairs (q, s) where s is an accepting state in $\mathcal{B}_{\neg\varphi}$
- Exploration algorithm: search for acceptance cycles
  - search for accepting pair that is reachable from itself
  - stack of search history can be used to produce counter-example
  - store set of already visited pairs (per search mode)

C. Courcoubetis, M. Vardi, P. Wolper, M. Yannakakis: *Memory-efficient algorithms for the verification of temporal properties.* Formal Methods in System Design 1:275–288 (1992)

Pseudo-Code

```
void check_ltl(KripkeStructure ks, Buchi aut) {
  Stack stack = new Stack();  Set visited = new Set();  Pair seed = null;

  void dfs(boolean cycle_mode) {
    Pair p = stack.top();
    if (cycle_mode && (p == seed)) { report acceptance cycle and exit }
    if (! visited.contains(p, cycle_mode) {
      visited.add(p, cycle_mode);
      foreach (Pair q in p.successors(ks, aut)) {
        stack.push(q);
        dfs(cycle_mode);
        if (! cycle_mode && aut.isAccepting(q)) {
          seed = q; dfs(true);
    } } }
    stack.pop();
  }
  // initialization
  foreach (Pair p in makeInitialPairs(ks, aut)) {
    stack.push(p); dfs(false);
} }
```

Optimizations

- Problem: state explosion
  - size of state space exponential in size of system description
  - main memory will be exhausted beyond $\sim 10^7$ states
  - disk storage is orders of magnitude slower than main memory
- Compression (of set *visited*)
  - store *signature* instead of full state $\rightsquigarrow$ hash conflicts
  - store only some states (at least one per loop), recompute others
- Reduction (of state space)
  - exploit *symmetries*: identify states up to equivalence relation
  - identify executions that differ only in order of *independent transitions*
- Abstraction (of transition system)
  - omit parts of state description that is "irrelevant"
  - automatically identify irrelevant system parts for given property