Temporal logics and explicit-state model checking

Pierre Wolper
Université de Liège

Topics to be covered

- Introducing explicit-state model checking
- Finite automata on infinite words
- Temporal Logics and their relation to automata
- Algorithms for explicit-state model checking

Introducing explicit-state model checking

- The system to be verified is modelled as a finite-state transition system, which is computed from the program describing the system.

- One explores this transition system state-by-state in order to check that it has the required properties:

  - simple state properties such as reachability,

  - properties of sequences of states expressed in a specification language: temporal logic.

- In practice, one needs to compute the *reachable* states of this transition system.

- To obtain good algorithms one proceeds by reducing the problem to a pure state reachability problem. For this, the temporal logic specifications are converted to finite automata.

Automata on infinite words: motivation

- Infinite behaviors play an important role in the analysis of reactive systems.

- Transition systems generating or recognizing infinite behaviors are useful in the analysis of reactive systems (fairness conditions and liveness properties).

- There exists a rich theory for such transition systems with a *finite* number of states: the theory of finite-automata on infinite words.

- In this theory, the objects recognized by the automata are infinite words defined over a finite alphabet.

Words and Automata

- A finite word of length n over an alphabet Σ is a mapping $w : \{0, \dots n-1\} \rightarrow \Sigma$
- An automaton on finite words is a description of a set of finite words (those that it accepts).
- An infinite word (ω-word) over an alphabet Σ is a mapping $w : \mathbf{N} \rightarrow \Sigma$
- An automaton on infinite words is a description of a set of infinite words (those that it accepts).

Note: Automata are often introduced as a model of computation. Here, we will just view them as a description of a set of words.

Automata on Infinite Words

Same structure as automata on finite words:

$A = \{\Sigma, S, \delta, S_0, F\}$ where

- Σ is an alphabet,
- S is a set of states,
- $\delta : S \times \Sigma \to S$ (deterministic) or $\delta : S \times \Sigma \to 2^S$ (nondeterministic) is a transition function,
- $S_0 \subseteq S$ is a set of initial states, and
- $F \subseteq S$ is a set of accepting states.

What changes is the semantics.

When does an Automaton accept a Word?

From now on, we consider the more general case of nondeterministic automata.

Finite words $w : \{0, \ldots n-1\} \rightarrow \Sigma$

The word w is accepted by the automaton if there is a labeling of the word by states

$$\rho : \{0, \ldots n\} \rightarrow S$$

such that

- $\rho(0) \in S_0$ (the initial label is an initial state),
- $\forall 0 \leq i \leq n-1$, $\rho(i+1) \in \delta(w(i), \rho(i))$ (the labeling is compatible with the transition relation),
- $\rho(n) \in F$ (the last label is an accepting state).

Don't think of automata as programs, but as a formalism for defining sets of words that has a very simple semantics.

Infinite words $w : \mathbf{N} \rightarrow \Sigma$

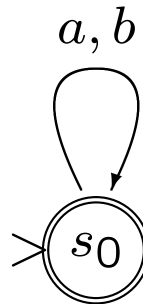
The word w is accepted by the automaton if there is a labeling of the word by states

$$\rho : \mathbf{N} \rightarrow S$$

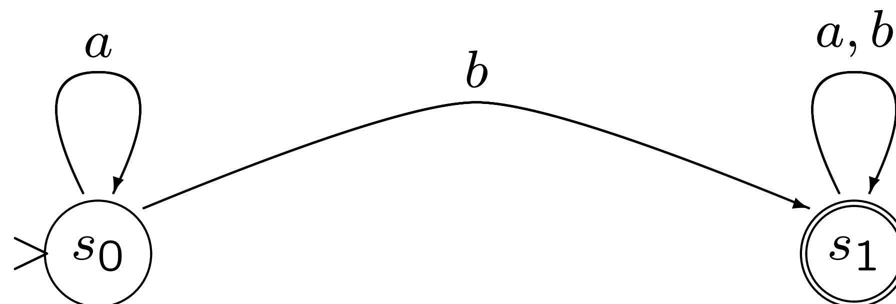
such that

- $\rho(0) \in S_0$ (the initial label is an initial state),
- $\forall 0 \leq i,\ \rho(i + 1) \in \delta(w(i), \rho(i))$ (the labeling is compatible with the transition relation),
- $inf(\rho) \cap F \neq \emptyset$ where $inf(\rho)$ is the set of states that appear infinitely often in ρ (the set of repeating states intersects F).

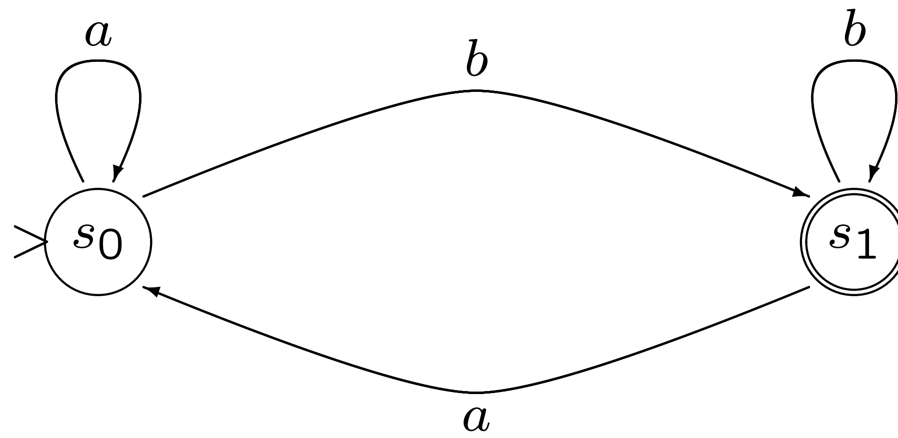
Examples



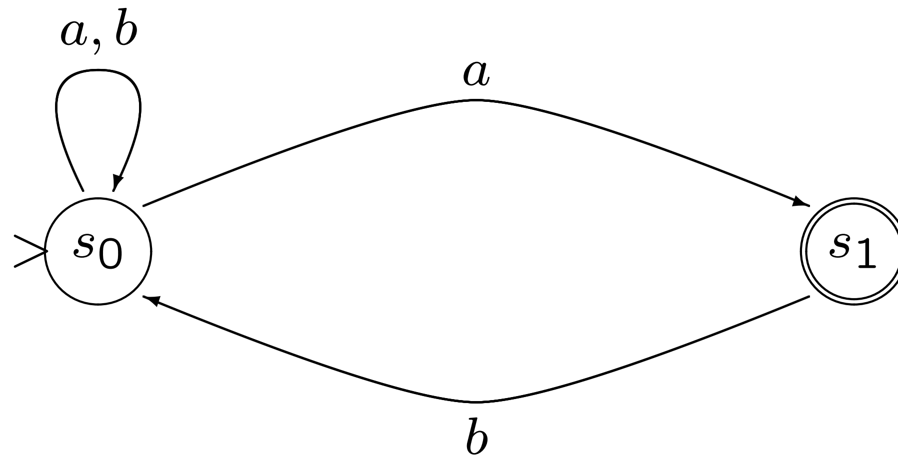
All words over $\Sigma = \{a, b\}$



An automaton accepting $a^\star b(a \cup b)^\omega$



All words containing b infinitely often



Infinitely often ab

Other types of Acceptance conditions

Büchi: $F \subseteq S$,
$\text{inf}(\rho) \cap F \neq \emptyset$.

Generalized Büchi: $\mathcal{F} \subseteq 2^S$,
i.e. $F = \{F_1, \ldots, F_m\}$
For each F_i, $\text{inf}(\rho) \cap F_i \neq \emptyset$.

Rabin: $F \subseteq 2^S \times 2^S$,
i.e. $F = \{(G_1, B_1), (G_2, B_2), \ldots, (G_m, B_m)\}$
For some pair $(G_i, B_i) \in F$, $\text{inf}(\rho) \cap G_i \neq \emptyset$ and $\text{inf}(\rho) \cap B_i = \emptyset$

For nondeterministic automata, all define the ω-regular languages :

$$\bigcup_i \alpha_i \beta_i^\omega$$

where α_i and β_i are finite-word regular languages and ω denotes infinite repetition.

For deterministic automata, Büchi and Generalized Büchi conditions are weaker.

Properties of Büchi Automata

- Closed under union, intersection, projection, and complementation.
- Nonemptiness easy to decide:
  - Check if some accepting state is accessible from an initial state and (nontrivially) from itself.
  - Linear time (compute the strongly connected components).
  - NLOGSPACE.
- For Rabin conditions, nonemptiness can also be decided in polynomial time. The simplest algorithms involve a translation to nondeterministic Büchi automata.
- For Street automata, the translation to nondeterministic Büchi automata is exponential.

Closure under Union

Given $A_1 = (\Sigma, S_1, \delta_1, S_{01}, F_1)$ and
$A_2 = (\Sigma, S_2, \delta_2, S_{02}, F_2)$,

the automaton $A = \{\Sigma, S, \delta, S_0, F\}$ where

- $S = S_1 \cup S_2, \quad S_0 = S_{01} \cup S_{02}, \quad F = F_1 \cup F_2,$
- $t \in \delta(s, a)$ if $\begin{cases} \text{either } t \in \delta_1(s, a) & \text{and } s \in S_1, \\ \text{or } t \in \delta_2(s, a) & \text{and } s \in S_2. \end{cases}$

accepts $L(A_1) \cup L(A_2)$.

Closure under Intersection

Given $A_1 = (\Sigma, S_1, \delta_1, S_{01}, F_1)$
and $A_2 = (\Sigma, S_2, \delta_2, S_{02}, F_2)$,

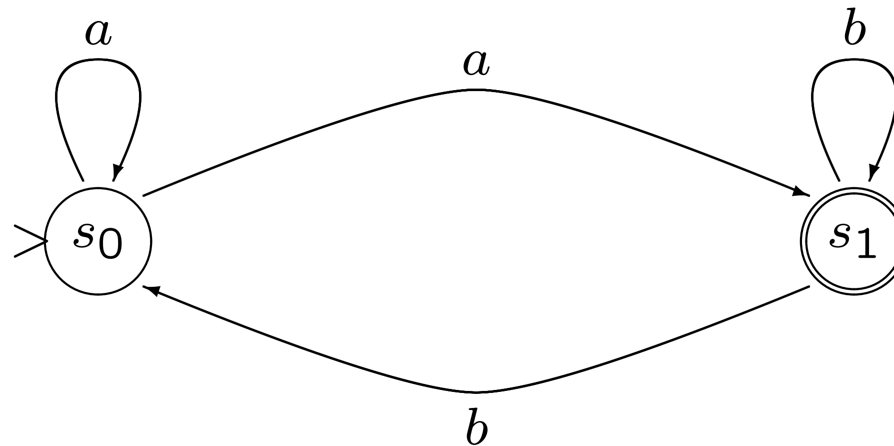
The generalized Büchi automaton and $A = \{\Sigma, S, \delta, S_0, \mathcal{F}\}$ where

- $S = S_1 \times S_2, \quad S_0 = S_{01} \times S_{02}$,
- $\mathcal{F} = \{F_1 \times S_2, S_1 \times F_2\}$,
- $(u, v) \in \delta((s, t), a)$ if $u \in \delta_1(s, a)$ and $v \in \delta_2(t, a)$.

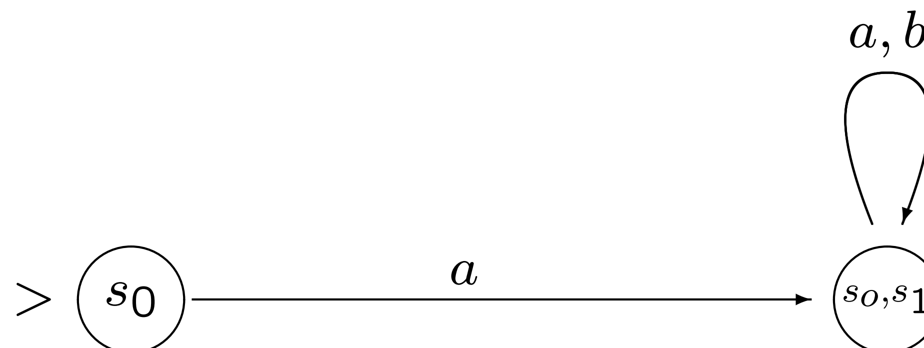
accepts the language $L(A_1) \cap L(A_2)$.

Projection is also simple, but *complementation* is completely nontrivial (the Rabin-Scott subset construction is not sufficient!).

Example:
Failure of the Rabin-Scott Construction



A simple Büchi Automaton



The Rabin-Scott construction applied to the automaton above

From Generalized Büchi to Büchi

Given $A = (\Sigma, S, \delta,\ S_0, \mathcal{F})$,
where $\mathcal{F} = \{F_1, \ldots, F_k\}$,

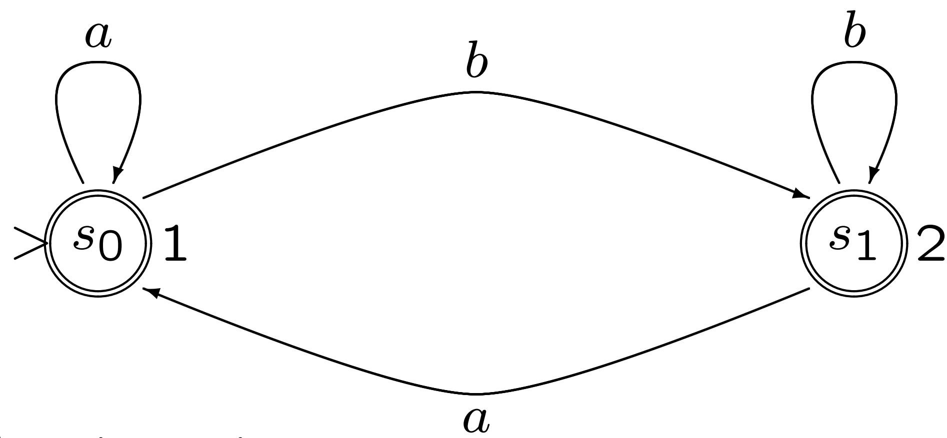
$A' = (\Sigma, S', \delta', S'_0, F')$ where

- $S' = S \times \{1, \ldots, k\}$.
- $S'_0 = S_0 \times \{1\}$.
- δ' is defined by $(t, i) \in \delta'((s, j), a)$ if

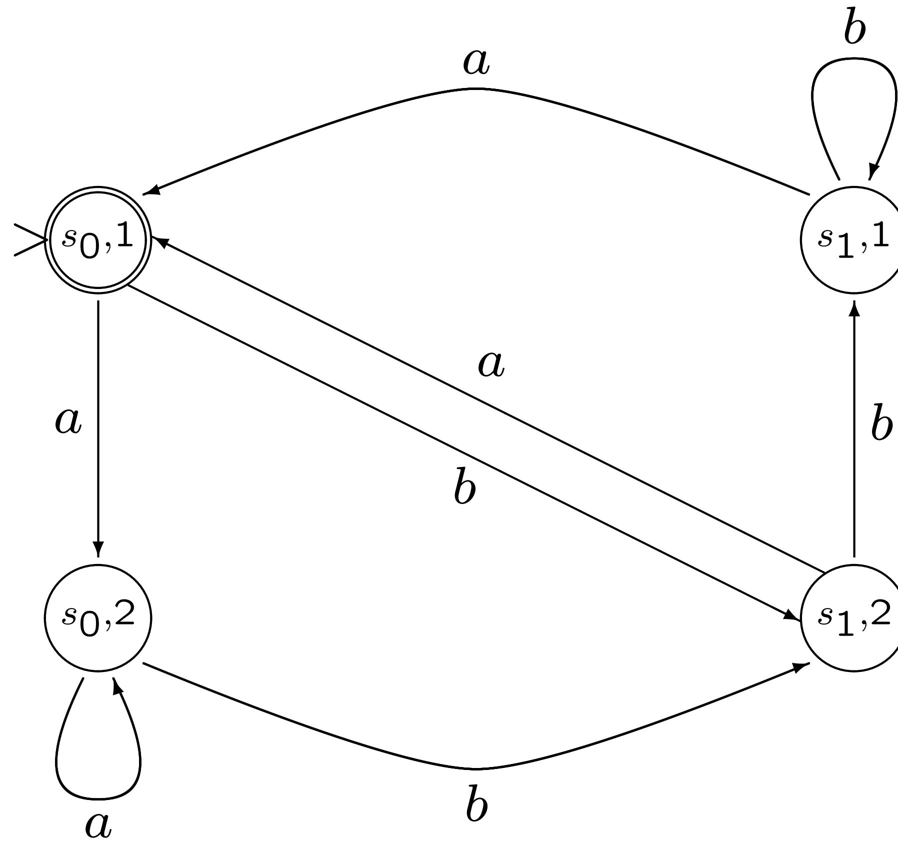
$$t \in \delta(s, a) \text{ and } \begin{cases} i = j & \text{if } s \notin F_j, \\ i = (j \bmod k) + 1 & \text{if } s \in F_j. \end{cases}$$

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

accepts $L(A)$.



A generalized Büchi automaton



From generalized Büchi to Büchi

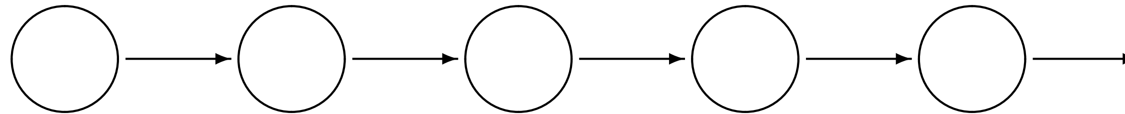
Temporal Logics and their Relation to Automata

A Language for Writing Specifications: Temporal Logic

- Language for describing properties of infinite sequences,
- Extension of propositional logic (or first-order logic),
- Uses *Temporal Operators* to describe temporal (sequencing) properties.

Temporal Logic: Interpretations

- Temporal logic is interpreted on sequences of states:



- each state in the sequence gives a truth value to atomic propositions,

- a formula is given a meaning in a state (world, interpretation) of a sequence,

- *temporal operators* indicates in which states the formula (or parts thereof) should be interpreted.

Temporal Operators

1. $\bigcirc$ which is read "at the next time" (in the next state of the sequence)

2. $\Box$ which is read "always in the future" (in all future states of the sequence)

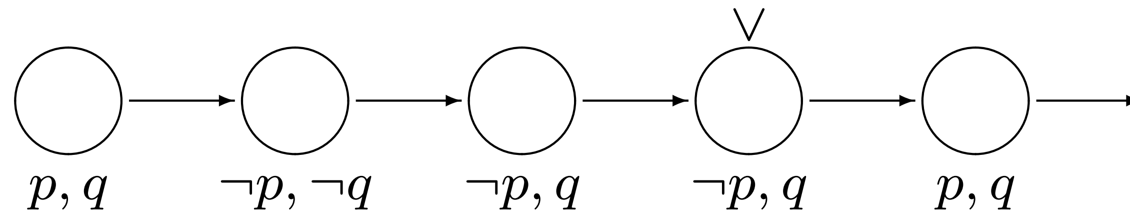
3. $\Diamond$ which is read "eventually" (in some future state of the sequence)

4. U which is read "until" (binary operator)

5. $\tilde{U}$ which is read "releases" (binary operator)

Examples

In the marked state of the sequence



the following formulas are *true*:

$$\neg p \wedge q$$

$$\bigcirc(p \wedge q)$$

the following formula is *false*:

$$\Box\, p$$

More Examples

- $\Box(p \supset \bigcirc q)$ is satisfied by all sequences in which each state where p is true is immediately followed by a state in which q is true.

- $\Box(p \supset \bigcirc(\neg q \, U \, r))$ is satisfied by all sequences such that if p is true in a state, then q remains false from the next state on and until the first state where r is true, which must occur.

- The formula $\Box \Diamond p$ is satisfied by all sequences in which p is true infinitely often in the future

- The formula $p \, \tilde{U} \, q$ is satisfied by all sequences in which q is always true unless this obligation is *released* by p being true in a previous state.

Formal Syntax of Temporal Logic

The formulas of linear-time temporal logic (LTL) built from a set of atomic propositions P are the following.

- $\mathbf{true}$, $\mathbf{false}$, p, or $\neg p$, for all $p \in P$;
- $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$, where φ_1 and φ_2 are LTL formulas;
- $\bigcirc \varphi_1$, $\varphi_1 \, U \, \varphi_2$, or $\varphi_1 \, \tilde{U} \, \varphi_2$, where φ_1 and φ_2 are LTL formulas.

Two operators are then defined as abbreviations:

- $\Diamond \varphi = \mathbf{true} \, U \, \varphi$ ("eventually").
- $\Box \varphi = \mathbf{false} \, \tilde{U} \, \varphi$ ("always").

Semantics of Temporal Logic

The semantics of LTL is defined with respect to paths $\pi : \mathbf{N} \rightarrow 2^P$. For a path π, π^i represents the suffix of π obtained by removing its i first states, i.e. $\pi^i(j) = \pi(i + j)$. The rules given the truth of a formula in the first state of a path π are the following.

- For all π, we have $\pi \models \mathbf{true}$ and $\pi \not\models \mathbf{false}$.
- $\pi \models p$ for $p \in P$ iff $p \in \pi(0)$.
- $\pi \models \neg p$ for $p \in P$ iff $p \not\in \pi(0)$.
- $\pi \models \varphi_1 \wedge \varphi_2$ iff $\pi \models \varphi_1$ and $\pi \models \varphi_2$.
- $\pi \models \varphi_1 \vee \varphi_2$ iff $\pi \models \varphi_1$ or $\pi \models \varphi_2$.
- $\pi \models \bigcirc \varphi_1$ iff $\pi^1 \models \varphi_1$.
- $\pi \models \varphi_1 \, U \, \varphi_2$ iff there exists $i \geq 0$ such that $\pi^i \models \varphi_2$ and for all $0 \leq j < i$, we have $\pi^j \models \varphi_1$.
- $\pi \models \varphi_1 \, \tilde{U} \, \varphi_2$ iff for all $i \geq 0$ such that $\pi^i \not\models \varphi_2$, there exists $0 \leq j < i$ such that $\pi^j \models \varphi_1$.

In the logic we have defined, negation is only applied to atomic propositions. This restriction can be lifted with the help of the following relations

$$\pi \not\models \varphi_1 \, U \, \varphi_2 \text{ iff } \pi \models (\neg\varphi_1) \, \tilde{U} \, (\neg\varphi_2)$$

$$\pi \not\models \varphi_1 \, \tilde{U} \, \varphi_2 \text{ iff } \pi \models (\neg\varphi_1) \, U \, (\neg\varphi_2)$$

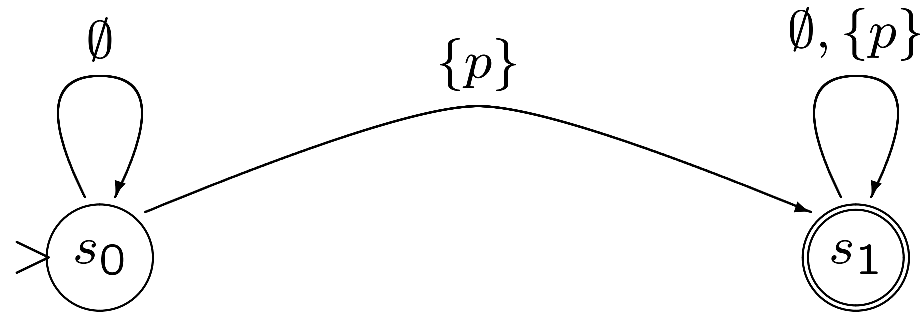
$$\pi \not\models \bigcirc \varphi_1 \text{ iff } \pi \models \bigcirc \neg\varphi_1$$

A linear time temporal logic formula is a description of a set of infinite sequences (those that satisfy it)

From LTL to Automata

Given a LTL formula φ built from a set of atomic propositions P, construct an automaton on infinite words over the alphabet 2^P that accepts exactly the infinite sequences satisfying φ.

Example:



An automaton for $\Diamond p$

Idea of the Construction

To determine if a sequence $\pi : \mathbf{N} \rightarrow 2^P$ satisfies a formula φ, one can proceed by labeling it with subformulas of φ in a way that is compatible with LTL semantics. The subformulas of φ are the *closure* of φ ($cl(\varphi)$).

- $\varphi \in cl(\varphi)$,
- $\varphi_1 \wedge \varphi_2 \in cl(\varphi) \Rightarrow \varphi_1, \varphi_2 \in cl(\varphi)$,
- $\varphi_1 \vee \varphi_2 \in cl(\varphi) \Rightarrow \varphi_1, \varphi_2 \in cl(\varphi)$,
- $\bigcirc \varphi_1 \in cl(\varphi) \Rightarrow \varphi_1 \in cl(\varphi)$,
- $\varphi_1 \, U \, \varphi_2 \in cl(\varphi) \Rightarrow \varphi_1, \varphi_2 \in cl(\varphi)$.
- $\varphi_1 \, \tilde{U} \, \varphi_2 \in cl(\varphi) \Rightarrow \varphi_1, \varphi_2 \in cl(\varphi)$.

$$cl(\Diamond \neg p) = \{\Diamond \neg p, \neg p\}$$

What are the labeling rules?

For every position $i \geq 0$ the labeling $\tau : \mathbf{N} \rightarrow 2^{cl(\varphi)}$ has to satisfy the following:

- For $p \in P$, if $p \in \tau(i)$ then $p \in \pi(i)$, and if $\neg p \in \tau(i)$ then $p \notin \pi(i)$
- if $\varphi_1 \wedge \varphi_2 \in \tau(i)$ then $\varphi_1 \in \tau(i)$ and $\varphi_2 \in \tau(i)$.
- if $\varphi_1 \vee \varphi_2 \in \tau(i)$ then $\varphi_1 \in \tau(i)$ or $\varphi_2 \in \tau(i)$.

This takes care of propositions and boolean connectives. What about the temporal operators?

Labeling Rules for Temporal Operators

Even though the temporal operators, have a semantics that looks at the whole sequence, it can (mostly) be reduced to local conditions.

$$
\begin{array}{rcl}
p\,U\,q & \equiv & (q \vee (p \wedge \bigcirc(p\,U\,q))) \\
p\,\tilde{U}\,q & \equiv & (q \wedge (p \vee \bigcirc(p\,\tilde{U}\,q)))
\end{array}
$$

We thus add the following labeling rules for all positions $i \geq 0$:

- if $\bigcirc\varphi_1 \in \tau(i)$ then $\varphi_1 \in \tau(i+1)$.
- if $\varphi_1\,U\,\varphi_2 \in \tau(i)$ then either $\varphi_2 \in \tau(i)$, or $\varphi_1 \in \tau(i)$ and $\varphi_1\,U\,\varphi_2 \in \tau(i+1)$.
- if $\varphi_1\,\tilde{U}\,\varphi_2 \in \tau(i)$ then $\varphi_2 \in \tau(i)$, and either $\varphi_1 \in \tau(i)$ or $\varphi_1\,\tilde{U}\,\varphi_2 \in \tau(i+1)$

There is only one missing condition that concerns formulas called *eventualities*

$$(\varphi_1 \, U \, \varphi_2) \;\equiv\; \varphi_1 \wedge (\varphi_2 \vee \bigcirc(\varphi_1 \, U \, \varphi_2))$$

φ_2, has to be true at some point!

Interpreting Labelings

A sequence π satisfies a formula φ if one can find a labeling τ satisfying

- The conditions above,
- $\varphi \in \tau(0)$, and
- If an eventuality formula $\varphi_1 \, U \, \varphi_2$ is in $\tau(i)$, then for some $j \geq i$, $\varphi_2 \in \tau(j)$.

An automaton on infinite words is just a labeling rule. Actually, we have defined an automaton on infinite words that accepts the models of a temporal logic formula!

The Automaton

It is a generalized Büchi automaton
$A_\varphi = (\Sigma, S, \delta, S_0, \mathcal{F})$ where

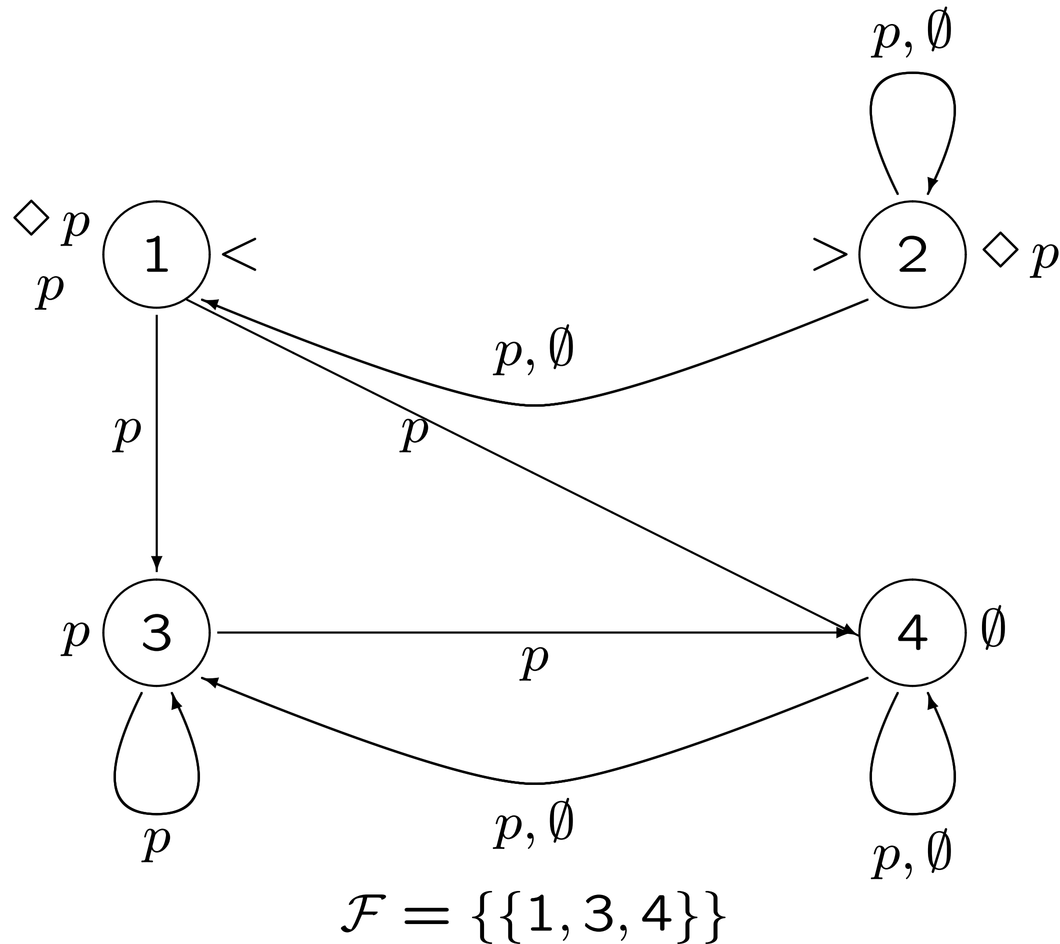
- $\Sigma = 2^P$,
- The set of states S is the set of possible labels, i.e. the subsets $\mathbf{s}$ of $2^{cl(\varphi)}$ that satisfy
  - if $\varphi_1 \wedge \varphi_2 \in \mathbf{s}$ then $\varphi_1 \in \mathbf{s}$ and $\varphi_2 \in \mathbf{s}$.
  - if $\varphi_1 \vee \varphi_2 \in \mathbf{s}$ then $\varphi_1 \in \mathbf{s}$ or $\varphi_2 \in \mathbf{s}$.

- The transition function δ checks that the propositional labeling matches the one in the sequence being considered and that the rules for the temporal operators are satisfied. Thus, $\mathbf{t} \in \delta(\mathbf{s}, \mathbf{a})$ iff
  - For all $p \in P$, if $p \in \mathbf{s}$ then $p \in \mathbf{a}$.
  - For all $p \in P$, if $\neg p \in \mathbf{s}$ then $p \notin \mathbf{a}$.
  - if $\bigcirc \varphi_1 \in \mathbf{s}$ then $\varphi_1 \in \mathbf{t}$.
  - if $\varphi_1 \, U \, \varphi_2 \in \mathbf{s}$ then either $\varphi_2 \in \mathbf{s}$, or $\varphi_1 \in \mathbf{s}$ and $\varphi_1 \, U \, \varphi_2 \in \mathbf{t}$.
  - if $\varphi_1 \, \tilde{U} \, \varphi_2 \in \mathbf{s}$ then $\varphi_2 \in \mathbf{s}$ and either $\varphi_1 \in \mathbf{s}$, or $\varphi_1 \, \tilde{U} \, \varphi_2 \in \mathbf{t}$.
- The initial states are those that contain φ, $S_0 = \{\mathbf{s} \in S \mid \varphi \in \mathbf{s}\}$.

- For the accepting condition, we need to impose that, for every eventuality formula $\varphi_1 \, U \, \varphi' \equiv e(\varphi') \in cl(\varphi)$, any state that contains that formula is followed by a state that contains φ'.
  - We have the property that, from every state in which $e(\varphi')$ appears, the transition relation ensures that this formula keeps appearing until the first state in which φ' appears.
  - It is thus sufficient to require that, for every eventuality $e(\varphi')$, one goes infinitely often either through a state in which it does not appear, or through a state in which the eventuality and φ' both appear.
  - The acceptance condition is thus a generalized Büchi condition. If the eventualities are $e_1(\varphi_1), \ldots e_m(\varphi_m)$, it is $\mathcal{F} = \{\Phi_1, \ldots, \Phi_m\}$ where $\Phi_i = \{\mathbf{s} \in S \mid e_i, \varphi_i \in \mathbf{s} \vee e_i \notin \mathbf{s}\}$.

Example

The automaton for $\Diamond p$



$\mathcal{F} = \{\{1, 3, 4\}\}$

Note: Transitions leading to states with more (nonatomic) elements of the closure than necessary were not included.

Improvements to the Construction

- Build the automaton incrementally from the formula φ by decomposing and expanding the temporal operators.
- Merge states which only differ by atomic propositions since these are determined by the labels of the transitions.
- When possible, detect inconsistent nodes without fully expanding to the propositional level.

Branching Time Temporal Logic

Branching time temporal logic are interpreted over *infinite trees* (varying degree) in which each node is a state assigning truth values to the atomic propositions.

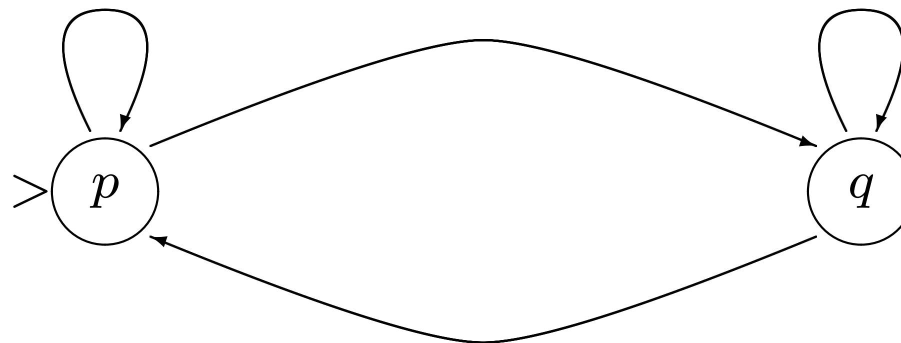
More generally, branching time temporal logic is interpreted over Kripke structures $K = \langle P, W, R, w^0, L \rangle$, where

- P is a set of atomic propositions,
- W is a set of states,
- $R \subseteq W \times W$ is a transition relation that must be total (i.e., for every $w \in W$ there exists $w' \in W$ such that $\langle w, w' \rangle \in R$),
- w^0 is an initial state, and
- $L : W \rightarrow 2^P$ maps each state to the set of atomic propositions true in that state.

From a state of a Kripke structure leave a number of *paths*, i.e. infinite sequence of states, $\pi = w_0, w_1, \ldots$ such that for every $i \geq 0$, $\langle w_i, w_{i+1} \rangle \in R$.

Note that by identifying $\pi(i)$ with $L(w_i)$, a path in a Kripke structure defines an LTL interpretation.

Example



Path Quantifiers

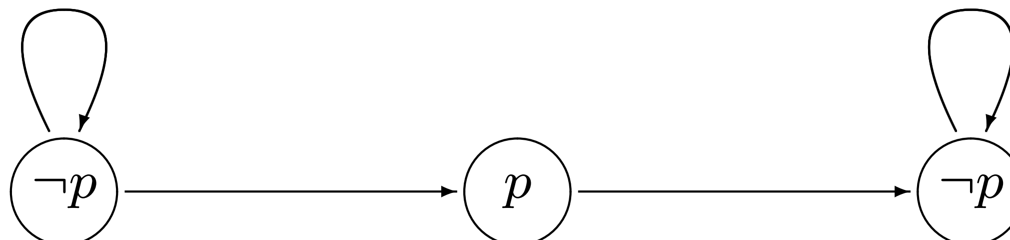
Branching-time temporal logic is linear-time temporal logic extended with *path quantifiers*:

- A for all paths,
- E for some paths.

Path quantifiers are applied to linear-time formulas which then become *state formulas* and can be used as atoms in a new path formula.

Examples

$A\Box p$, $E\Box p$, $A(p \supset \bigcirc E\Box p)$, $A\Box E\bigcirc p$, $A(\Box\Diamond p)$, $E(\Box\Diamond p)$, $E\Box E\Diamond p$.



Formal Syntax of Branching-time Temporal Logic

Syntax
We first define the more general logic CTL*. A CTL* state formula is either:

- **true**, **false**, p, or $\neg p$, for all $p \in P$;
- $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$, where φ_1 and φ_2 are CTL* state formulas;
- $A\psi$ or $E\psi$, where ψ is a CTL* path formula.

A CTL* path formula is either:

- A CTL* state formula;
- $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$, $\bigcirc \psi_1$, $\psi_1 U \psi_2$, or $\psi_1 \tilde{U} \psi_2$, where ψ_1 and ψ_2 are CTL* path formulas.

CTL* is the set of state formulas generated by the above rules.

A restricted form of Branching-time Temporal Logic

The logic *CTL* is a restricted subset of CTL* in which the temporal operators must be immediately preceded by a path quantifier.

Formally, it is the subset of CTL* obtained by defining the path formulas as follows.

A CTL path formula is of the form

- $\bigcirc \varphi_1$, $\varphi_1 \, U \, \varphi_2$, or $\varphi_1 \, \tilde{U} \, \varphi_2$, where φ_1 and φ_2 are CTL state formulas.

Examples

- CTL formulas: $A \Box p$, $E \Box p$, $A \Box E \bigcirc p$, $E \Box E \Diamond p$.
- CTL* formulas: $A(p \supset \bigcirc E \Box p)$, $A(\Box \Diamond p)$, $E(\Box \Diamond p)$.

Semantics

Given a Kripke structure $K = \langle P, W, R, w^0, L \rangle$, the truth of a CTL* formula in a state w is defined as follows.

- For all w, we have $w \models \mathbf{true}$ and $w \not\models \mathbf{false}$.
- $w \models p$ for $p \in P$ iff $p \in L(w)$.
- $w \models \neg p$ for $p \in P$ iff $p \notin L(w)$.
- $w \models \varphi_1 \wedge \varphi_2$ iff $w \models \varphi_1$ and $w \models \varphi_2$.
- $w \models \varphi_1 \vee \varphi_2$ iff $w \models \varphi_1$ or $w \models \varphi_2$.
- $w \models A\psi$ iff for every path $\pi = w_0, w_1, \ldots$, with $w_0 = w$, we have $\pi \models \psi$.
- $w \models E\psi$ iff there exists a path $\pi = w_0, w_1, \ldots$, with $w_0 = w$, such that $\pi \models \psi$.

- $\pi \models \varphi$ for a state formula φ, iff $w_0 \models \varphi$ where $\pi = w_0, w_1, \ldots$
- $\pi \models \psi_1 \wedge \psi_2$ iff $\pi \models \psi_1$ and $\pi \models \psi_2$.
- $\pi \models \psi_1 \vee \psi_2$ iff $\pi \models \psi_1$ or $\pi \models \psi_2$.
- $\pi \models \bigcirc \psi$ iff $\pi^1 \models \psi$.
- $\pi \models \psi_1 \, U \, \psi_2$ iff there exists $i \geq 0$ such that $\pi^i \models \psi_2$ and for all $0 \leq j < i$, we have $\pi^j \models \psi_1$.
- $\pi \models \psi_1 \, \tilde{U} \, \psi_2$ iff for all $i \geq 0$ such that $\pi^i \not\models \psi_2$, there exists $0 \leq j < i$ such that $\pi^j \models \psi_1$.

Algorithms for explicit-state model checking

The Verification Approach

- The central part of the verification process is to compute the transition system corresponding to the Formal Concurrent System to be analyzed.

- In practice, one needs to compute the *reachable* states of this transition system.

- One can then check properties by examing this transition systems, e.g. the reachability of a given state, the absence of deadlocks, other properties of all reachable states . . .

- The main problem is that the transition system can be extremely large. One thus needs techniques to optimize the computation of its reachable states.

Computing the Reachable States of a System: Depth-First Search

The following algorithm computes the transition system $ST = \{\Sigma, S, s_0, T\}$ corresponding to a formal concurrent system $FCS = \{\mathcal{P}, \mathcal{M}, \mathcal{T}\}$.

This algorithm uses a stack ($Stack$) and a data structure H used to memorize the set of visited states. This structure is usually a hash table.

The function enabled(s) returns the transitions of $\mathcal{T}$ that are enabled in a state $s \in S$. The function succ(s, t) where $s \in S$ and $t \in \mathcal{T}$ returns the state s' such that $(s, n(t), s') \in T$

```
procedure DFS()
    begin
        s := top(Stack);
        Tr := enabled(s)
        for all t ∈ Tr
            begin
            s' := succ(s, t)
            if s' ∉ H then
              begin
              insert s' in H;
              push s' onto Stack;
              DFS();
              end
            end
        pop(Stack);
    end
Stack := [s0]; H := {s0};
DFS();
```

Using Depth-First Search

- The depth-first algorithm makes it possible to check properties of the reachable states: invariants, reachability of a state with a given property, absence of deadlock, . . .
- If an erroneous state is reached, the stack contains an execution trace that makes this state reachable. This is very helpful for understanding why such a state can be reached.
- By adding an *observer process*, more elaborate properties can be checked and in particular properties expressed in linear-time temporal logic.
- When performing the depth-first search, the transition relation need not be stored. One refers to this by saying the verification is done **on-the-fly** (while exploring the transition system)

Checking Linear-Time Temporal Logic Properties

- To check that a transition system satisfies a linear-time temporal logic property, one needs to check that *all* its infinite behaviors satisfy this property.
- One checks that all these behaviors are *models* of the property. This procedure is thus called *model-checking*.
- To obtain a model checking procedure, the idea is to use the automaton on infinite words for the property as an observer for the system.
- However, this automaton is nondeterministic and it accepts a behavior is *there exists* some computation for which it accepts.
- Checking that *for all* behaviors of the transition system, *there exists* some accepting computation of the observer is problematic.
- The trick is to first complement the property to be checked.

A Model-Checking Algorithm for LTL

- One is given a formal concurrent system FCS and a LTL property f.
- The first step is to build an automaton on infinite words $A_{\neg f}$ that accepts exactly all infinite sequences that *do not* satisfy f.
- $A_{\neg f}$ is then added as an observer to FCS. If needed, observers for the necessary fairness conditions are added.
- The corresponding transition system is then computed by a depth-first search. It is a transition system with a generalized Büchi acceptance condition.

- One then checks whether this transition system is empty of not (accepts at least one infinite word). Any accepted infinite word is a behavior of the system that does not satisfy the property to be checked.

- Remaining problem: how does one check emptiness of a Generalized Büchi automaton by a depth-first search, i.e. without applying the more complicated strongly connected components algorithm?

Complexity of LTL Model Checking:
Linear in the size of the transition system (NLOGSPACE),
Exponential in the size of the FCS (PSPACE),
Exponential in the size of the LTL property (PSPACE).

A Simple Case for Model-Checking : Safety Properties

- Imagine a property f is such that the corresponding automaton has an empty acceptance condition. This should make model checking simpler.
- No acceptance condition, means that the only way a word cannot be accepted is by reaching a situation in which no transition is possible.
- Furthermore, since there is no acceptance condition, the Rabin-Scott subset construction can be used to determinize and complement such an automaton on infinite words.

- The complement automaton would then have a single accepting state from which all behaviors are accepted. Once this state is reached, the word is accepted whatever happens next. The automaton for $\neg f$ is thus really an automaton on *finite-words*.

- This has two consequences on the algorithmic aspects of checking safety properties:

  - Fairness conditions are not necessary.
  - The emptiness test is a simple reachability test.

Safety Properties: An Abstract Definition

- We work in the context in which a property is a set of infinite words over an alphabet Σ.
- A property P is thus a set $P \subseteq \Sigma^\omega$.
- Given two words $w_1, w_2 \in \Sigma^\omega$, let $\text{lcp}(w_1, w_2)$ denote the length of their longest common prefix. Define then the *distance* between two words $w_1, w_2 \in \Sigma^\omega$ as

$$d(w_1, w_2) = \frac{1}{\text{lcp}(w_1, w_2) + 1}$$

  This distance induces a topology on Σ^ω.
- An *open set* O is one in which each element is surrounded by a neighborhood that is in the set:

$$\text{if } w \in O, \text{ then } \exists \varepsilon \forall w' [d(w, w') \leq \varepsilon \supset w' \in O]$$

- A *closed set* is the complement of an open set. Note that a closed set contains the limit of its converging sequences of words.
- A **safety property** is a **closed** subset of Σ^ω.

Another Particular Class of Properties: Liveness Properties

- An automaton without an accepting condition defines a safety property. What class of properties is defined by automata that "only have an accepting condition"?

- Formally, a language L is in this class if it satisfies

$$(\forall w \in \Sigma^*)(\exists w' \in \Sigma^\omega) ww' \in L$$

- Topologically, a property $L \subseteq \Sigma^\omega$ is a liveness property if it is a **dense** set:

$$(\forall w \in \Sigma^\omega)\forall\varepsilon(\exists w' \in L)d(w, w') \leq \varepsilon$$

- It is possible to prove that every property is the intersection of a liveness and of a safety property.

Checking the Nonemptiness of (Generalized) Büchi Automata With a Depth-First Search

- A Büchi automaton accepts if some state in F is reachable from the initial state and from itself by a path of length ≥ 1 (For generalized Büchi automata, convert to Büchi).

- To check that it is nonempty, proceed as follows.

  1. Do a depth-first search of the reachable states.
  2. When doing this search, build a postorder list of reachable accepting states. Let this list be

     $$Q = f_1, \ldots, f_k$$

     where f_1 is the first postorder reachable accepting state and f_k is the last.
  3. Do a second depth-first search from the elements in Q.

The Second Search

1. Start the search from f_1.

2. Once the search from f_i is finished (f_i reached or no more reachable nodes) :

   (a) Restart the search from f_{i+1}, but

   (b) do not reconsider nodes that have been visited during searches from f_j, $j \leq i$.

3. Each node in the graph is only visited once.

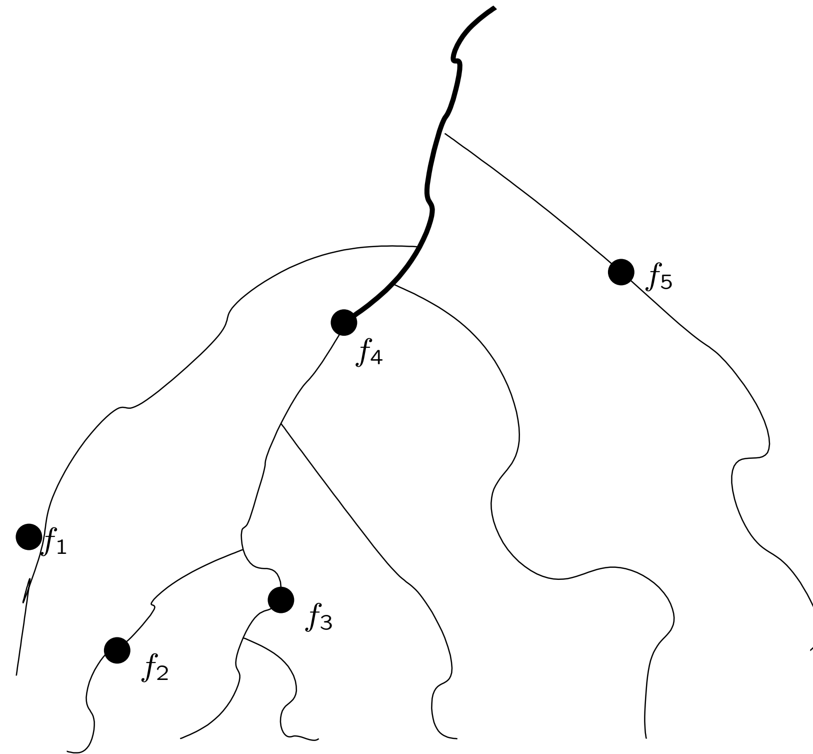
Note that the first and second search can be interleaved.

Why Does it Work?

1. If f_i is reachable from f_j, where f_j appears before f_i in postorder, then f_j is reachable from itself.

   - In postorder, the ancestors of a node separate the nodes preceding that node from the nodes following it.
   - Thus a path from a node A to a node following A in postorder must go through an ancestor of A and A is reachable form itself.

2. So, for the first f_i reachable from itself, the path p from f_i to itself cannot contain any node reachable from a previous f_j (if it did, there would be a path from f_j to f_i and f_j would be reachable from itself)


f_1
f_2
f_3
f_4
f_5

Algorithms for Checking Temporal Logic Properties (Branching Time)

The Logic CTL

Recall that CTL is the branching-time temporal logic whose formulas are defined as follows.

A CTL state formula is either:

- $\mathbf{true}$, $\mathbf{false}$, p, or $\neg p$, for all $p \in P$;
- $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$, where φ_1 and φ_2 are CTL state formulas;
- $A\psi$ or $E\psi$, where ψ is a CTL path formula.

A CTL path formula is of the form

- $\bigcirc \varphi_1$, $\varphi_1 \, U \, \varphi_2$, or $\varphi_1 \, \tilde{U} \, \varphi_2$, where φ_1 and φ_2 are CTL state formulas.

Examples
$A \Box p$, $E \Box p$, $A \Box E \bigcirc p$, $E \Box E \Diamond p$, $A(E(\Box p) \, \tilde{U} \, A(q \, U \, p))$.

Branching-Time Model Checking
The Problem

- One is given a finite transition system obtained from a concurrent program, i.e a finite Kripke structures $K = \langle P, W, R, w^0, L \rangle$, where
  - P is a set of atomic propositions,
  - W is a set of states,
  - $R \subseteq W \times W$ is a transition relation that must be total (i.e., for every $w \in W$ there exists $w' \in W$ such that $\langle w, w' \rangle \in R$),
  - w^0 is an initial state, and
  - $L : W \to 2^P$ maps each state to the set of atomic propositions true in that state.

- One is also given a CTL formula φ.

- The model-checking problem is to check that K satisfies (is a model of) φ.

- Note that in contrast to the linear-time case where there was an implicit "for all behaviors" quantification in the definition of the model-checking problem, there is no implicit quantification in the branching-time model-checking problem.

A First Approach

The problem can be solved by checking if the initial state w^0 of the Kripke structure can be labeled by φ while respecting the semantic rules of the logic. Formally, one checks whether there is a labeling $\tau : W \rightarrow cl(\varphi)$ such that the following rules are satisfied.

- $\varphi \in \tau(w^0)$
- For $p \in P$, if $p \in \tau(w)$ then $p \in L(w)$, and if $\neg p \in \tau(w)$ then $p \notin L(w)$
- if $\varphi_1 \wedge \varphi_2 \in \tau(w)$ then $\varphi_1 \in \tau(w)$ and $\varphi_2 \in \tau(w)$.
- if $\varphi_1 \vee \varphi_2 \in \tau(w)$ then $\varphi_1 \in \tau(w)$ or $\varphi_2 \in \tau(w)$.

- if $A \bigcirc \varphi_1 \in \tau(w)$ then for all w' such that $(w, w') \in R$, $\varphi_1 \in \tau(w')$.
- if $E \bigcirc \varphi_1 \in \tau(w)$ then for some w' such that $(w, w') \in R$, $\varphi_1 \in \tau(w')$.
- if $A(\varphi_1 \, U \, \varphi_2) \in \tau(w)$ then either $\varphi_2 \in \tau(w)$, or $\varphi_1 \in \tau(w)$ and for all w' such that $(w, w') \in R$ $A(\varphi_1 \, U \, \varphi_2) \in \tau(w')$.
- if $E(\varphi_1 \, U \, \varphi_2) \in \tau(w)$ then either $\varphi_2 \in \tau(w)$, or $\varphi_1 \in \tau(w)$ and for some w' such that $(w, w') \in R$ $E(\varphi_1 \, U \, \varphi_2) \in \tau(w')$.
- if $A(\varphi_1 \, \tilde{U} \, \varphi_2) \in \tau(w)$ then $\varphi_2 \in \tau(w)$, and either $\varphi_1 \in \tau(w)$ or for all w' such that $(w, w') \in R$ $A(\varphi_1 \, \tilde{U} \, \varphi_2) \in \tau(w')$
- if $E(\varphi_1 \, \tilde{U} \, \varphi_2) \in \tau(w)$ then $\varphi_2 \in \tau(w)$, and either $\varphi_1 \in \tau(w)$ or for some w' such that $(w, w') \in R$ $E(\varphi_1 \, \tilde{U} \, \varphi_2) \in \tau(w')$

Furthermore, to these rules we need to add the fact that the eventualities are satisfied.

Turning the Approach into an Algorithm

To turn the labeling rules above into an algorithm, we proceeed by labeling the structure with elements of the closure in an order compatible with the "subformula" order, starting with the innermost subformulas (i.e., the atomic or negated atomic propositions).

The general structure of the algorithm is thus the following.

1. Pick an formula $\varphi_i \in cl(\varphi)$ all subformulas of which have already been processed (initially this can only be an atomic or negated atomic proposition).
2. For each state $w \in W$, label w by φ_i if the labeling rules allow it (note that we label every time it is allowed, i.e. we label maximally).
3. Repeat until φ is processed.

We still need to describe how to do step 2.

- For formulas that are of the form $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $A \bigcirc \varphi_1$, or $E \bigcirc \varphi_1$, it is perfectly straightforward: one only needs to look at the current labeling.
- For formulas of the form $A(\varphi_1 \, U \, \varphi_2)$, $E(\varphi_1 \, U \, \varphi_2)$, $A(\varphi_1 \, \tilde{U} \, \varphi_2)$, or $E(\varphi_1 \, \tilde{U} \, \varphi_2)$, more needs to be done.

Coping with U and $\tilde{U}$ Formulas

For U and $\tilde{U}$ formulas, the labeling rules are recursive. Furthermore, for U formulas we have to guarantee that their second argument is indeed eventually satisfied.

Consider first the case of $A(\varphi_1\, U\, \varphi_2)$ formulas. We use the following observations.

- A node w that is already labeled by φ_2 can be unconditionally labeled by $A(\varphi_1\, U\, \varphi_2)$.
- If for all w' such that $(w, w') \in R$, w' is unconditionally labeled by $A(\varphi_1\, U\, \varphi_2)$ and if w is labeled by φ_1, then w can be unconditionally labeled by $A(\varphi_1\, U\, \varphi_2)$.

These observations can be directly translated into the following algorithm.

1. Label all nodes already labeled by φ_2 by $A(\varphi_1 \, U \, \varphi_2)$.
2. For each node $w \in W$, if all w' such that $(w, w') \in R$ are already labeled by $A(\varphi_1 \, U \, \varphi_2)$ and if w is labeled by φ_1, then label w by $A(\varphi_1 \, U \, \varphi_2)$.
3. Repeat step 2 until no more nodes need to be labeled.

For formulas of the form $A(\varphi_1 \tilde{U} \varphi_2)$, the situation is dual. Precisely, we start with an optimistic labeling (all states in which φ_2 appears) and then remove nodes that have been labeled unsoundly.

The rules are the following.

- A node w that is already labeled by φ_2 can be provisionally labeled by $A(\varphi_1 \tilde{U} \varphi_2)$.
- if a node w is not labeled φ_1 and some node w' such that $(w, w') \in R$ is not provisionally labeled by $A(\varphi_1 \tilde{U} \varphi_2)$, then remove $A(\varphi_1 \tilde{U} \varphi_2)$ from the label of w.

The algorithm is then the following.

1. Label all nodes already labeled by φ_2 by $A(\varphi_1 \tilde{U} \varphi_2)$.
2. For each node $w \in W$, if w is not labeled by φ_1 and if some w' such that $(w, w') \in R$ is not labeled by $A(\varphi_1 \tilde{U} \varphi_2)$, then remove $A(\varphi_1 \tilde{U} \varphi_2)$ from the label of w.
3. Repeat step 2 until no more nodes need to be unlabeled.

A very similar procedure can be used for formulas of the form $E(\varphi_1 \tilde{U} \varphi_2)$.

Labeling Rules and Fixpoints

The labeling algorithms we have just given for U and $\tilde{U}$ can be seen as the computation of *fixpoints.*

- Consider the set of subsets of W (2^W) ordered with the subset relation.
- Consider then a function $\mathcal{T} : 2^W \rightarrow 2^W$. A *fixpoint* of $\mathcal{T}$ is a set X such that $X = \mathcal{T}(X)$.
- The *least fixpoint* $\mu X.\mathcal{T}(X)$ of $\mathcal{T}$ is the smallest set such that $X = \mathcal{T}(X)$.
- The *greatest fixpoint* $\nu X.\mathcal{T}(X)$ of $\mathcal{T}$ is the largest set such that $X = \mathcal{T}(X)$.

Theorem. If $\mathcal{T}$ is monotonic, i.e. if when $X_1 \leq X_2$ then $\mathcal{T}(X_1) \leq \mathcal{T}(X_2)$, then least and greatest fixpoints exist. Furthermore, under continuity conditions (satisfied when W is finite), the fixpoints can be computed as follows (these computations terminate since W is finite).

$$\begin{array}{rcl} \mu X \mathcal{T}.(X) & = & \bigcup_i \mathcal{T}^i(\emptyset) \\ \nu X \mathcal{T}.(X) & = & \bigcap_i \mathcal{T}^i(W) \end{array}$$

Using these notions, we can write:

$$\begin{array}{rcl} A(\varphi_1 \, U \, \varphi_2) & = & \mu X(\varphi_2 \vee (\varphi_1 \wedge A \bigcirc X)) \\ A(\varphi_1 \, \tilde{U} \, \varphi_2) & = & \nu X(\varphi_2 \wedge (\varphi_1 \vee A \bigcirc X)) \end{array}$$

The algorithms we have given perform a computation corresponding exactly to the evaluation of these fixpoints.

Complexity Issues

The algorithms given for labeling with U and $\tilde{U}$ formulas have a complexity that is $O(|W|^2)$. This can be reduced to $O(|W|)$ by avoiding duplicate work.

Note that to achieve this complexity, we have to mark in one pass over the structure *all* nodes in which the formula is true.

For AU formulas, one uses a depth-first search and marks processed states (using a table *marked*).

The procedure then applies the recursive marking procedure *aulabel* to all nodes that have not yet been processed while handling a previous node:

for all $w \in W$
    begin
    if $\neg$marked(w) **then** aulabel(w, f)
    end

The two arguments of the procedure *aulabel* are a node w and a formula f of the form $A(\varphi_1 U \varphi_2)$.

```
procedure aulabel(w, f): boolean
  begin
    if marked(w) then
                   if f ∈ τ(w) then return(T)
                                else return(F);
    marked(w) := T;
    if φ2 ∈ τ(w) then begin
                        τ(w) := τ(w) ∪ {f};
                        return(T)
                      end;
    if φ1 ∉ τ(w) then return(F);
    for all w'.(w, w') ∈ R
      begin
      if ¬aulabel(w', f) then return(F)
      end;
      τ(w) := τ(w) ∪ {f};
      return(T)
  end
```

For $E\tilde{U}$ formulas, the algorithm is similar and is given below

```
for all w ∈ W
    begin
    if ¬marked(w) then eũlabel(w, f)
    end



procedure eũlabel(w, f): boolean
   begin
      if marked(w) then
                      if f ∈ τ(w) then return(T)
                                  else return(F);
      marked(w) := T;
      τ(w) := τ(w) ∪ {f};
      if φ2 ∉ τ(w) then begin
                          τ(w) := τ(w) \ {f};
                          return(F)
                          end;
```

if $\varphi_1 \in \tau(w)$ **then return**(T);
for all $w'.(w, w') \in R$
  begin
  if eũlabel(w', f) **then return**(T)
  end;
  $\tau(w) := \tau(w) \setminus \{f\}$;
  return(F)
end

Note that when these algorithms find a cycle that goes back to a node whose processing has started but is not finished, this node is assumed to be unlabeled by f in the case of an AU formula, and is assumed to be labeled by f in the case of an $E\tilde{U}$ formulas.

Indeed, for an AU formula, a cycle in which φ_2 does not occur has been found and the formula is false. For an $E\tilde{U}$ formula, a cycle in which φ_2 is always true has been found and the formula is thus true.

A similar approach cannot work for EU (or $A\tilde{U}$ formulas). Indeed, finding a cycle on which φ_2 is not true tells us nothing about and EU formula since only the existence of a satisfying path is required. The same applies dually to $A\tilde{U}$ formulas.

A different approach is thus needed for EU and $A\tilde{U}$ formulas. It consists of proceeding backwards through the graph from nodes whose marking is immediate.

For EU formulas, the algorithm is as below.

```
for all w ∈ W
    begin
    if φ2 ∈ τ(w) ∧ f ∉ τ(w) then eulabel(w, f)
    end
```

```
procedure eulabel(w, f)
   begin
      τ(w) := τ(w) ∪ {f};
      for all w'.(w', w) ∈ R
          begin
          if φ1 ∈ τ(w') ∧ f ∉ τ(w') then
                                        eulabel(w', f)
          end
      end
```

For $A\tilde{U}$ formulas, the algorithm is similar, except that we unlabel nodes.

```
for all w ∈ W
    begin
    τ(w) := τ(w) ∪ {f}
    end;

for all w ∈ W
    begin
    if φ2 ∉ τ(w) ∧ f ∈ τ(w) then aũlabel(w, f)
    end


procedure aũlabel(w, f)
   begin
      τ(w) := τ(w) \ {f};
      for all w′.(w′, w) ∈ R
          begin
          if φ1 ∉ τ(w′) ∧ f ∈ τ(w′) then
                                          aũlabel(w′, f)
          end
      end
```

Complexity Results

Since the model-checking algorithm uses time $O(|W|)$ for each element of the closure of the formula, its overall complexity is

$$O(|W| \times |\varphi|).$$

To summarize the situation is the following.

Complexity of LTL Model Checking:
Linear in the size of the transition system (NLOGSPACE),
Exponential in the size of the FCS (PSPACE),
Linear in the size of the CTL property.

Note that the only difference with respect to LTL model checking is the complexity in the size of the formula, which usually is not the crucial factor.