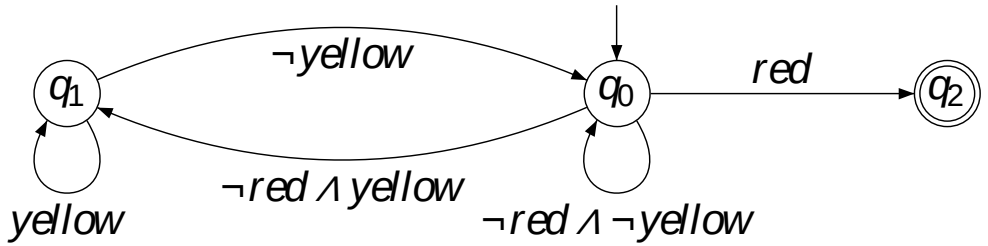
Model Checking finite transitions systems over linear time properties

MODEL CHECKING *SAFETY* PROPERTIES

Regular safety properties

- A safety property can be defined by the set of finite words («bad prefixes») over the «alphabet» 2^{AP} *violating* the property
- A safety property P_{safe} over AP is called **regular** if its set of bad prefixes is a *regular language* over 2^{AP}

  - Regular safety properties can be defined by a Nondeterm. Finite Automaton recognizing finite words in $(2^{AP})^*$



Model checking safety properites

- Let P_{safe} be a regular safety property over the atomic propositions AP
- Let TS be a finite transition system without terminal states
- Model checking: verify if $TS \models P_{safe}$

- Let A be an NFA recognizing the (minimal) bad prefixes of Psafe.

- Define an invariant $P_{inv(A)} = \bigwedge_{q \in F} \neg\, q$
  - Meaning: a propositional formula stating that A is not in a final state

Reduced to Invariant checking

1) Build a product of transition system TS and NFA A: TS $\otimes$ A, similar to the synchronous product of two NFA.
2) Then the following statements are equivalent:

$$(a)\quad TS \models P_{safe}$$

$$(b)\quad Traces_{fin}(TS) \cap L(A) = \emptyset$$

$$(c)\quad TS \otimes A \models P_{inv(A)}$$

Hence, *the verification of a regular safety property is reduced to invariant checking* ($P_{inv(A)}$ is the propositional formula stating that A is not in a final state)

Product of TS and NFA

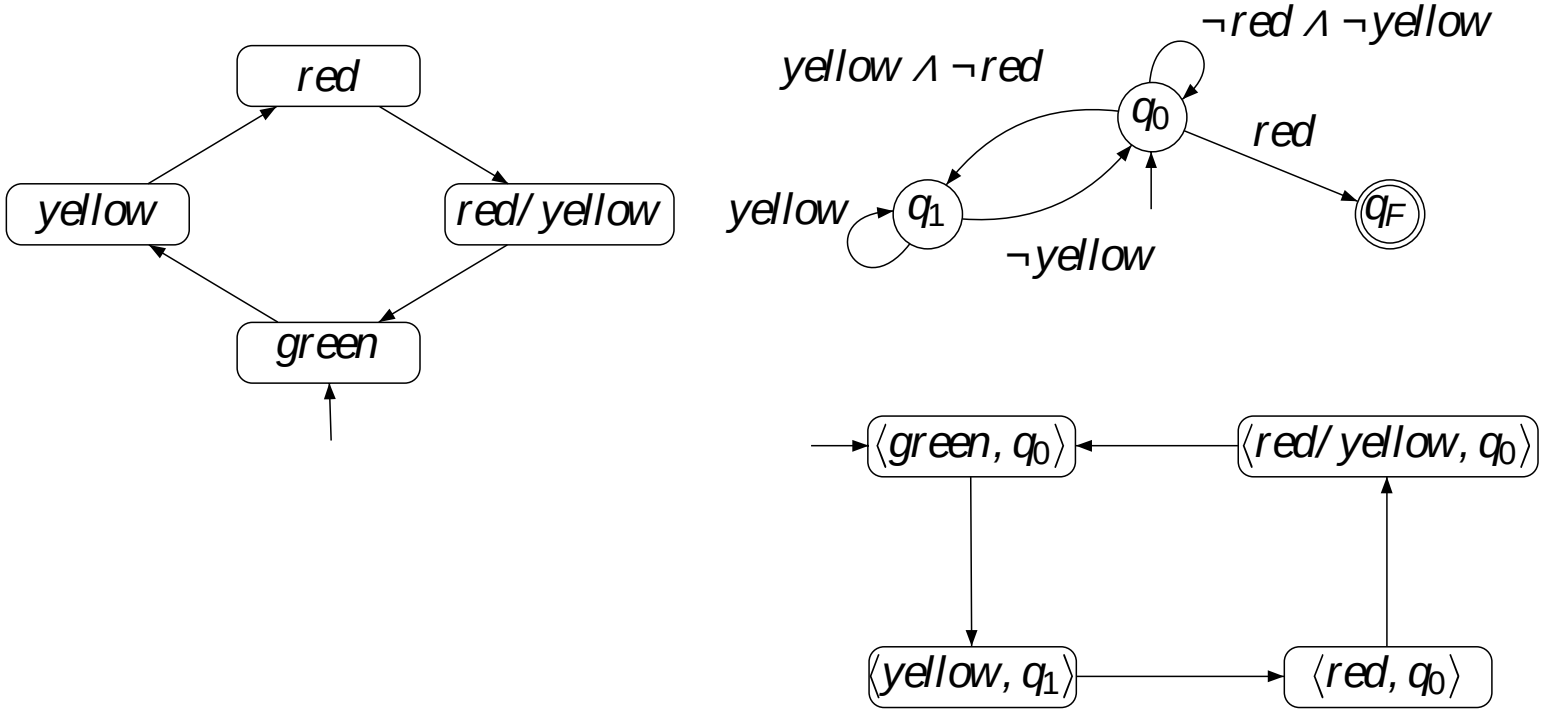


Figure 4.6: German traffic light (left upper figure), an NFA (right upper figure), and their product (lower figure).

Algorithm and complexity

Algorithm 5 Model-checking algorithm for regular safety properties

Input: finite transition system TS and regular safety property P_{safe}
Output: true if $TS \models P_{safe}$. Otherwise false plus a counterexample for P_{safe}.

Let NFA $\mathcal{A}$ (with accept states F) be such that $\mathcal{L}(\mathcal{A})$ = bad prefixes of P_{safe}
Construct the product transition system $TS \otimes \mathcal{A}$
Check the invariant $P_{inv(\mathcal{A})}$ with proposition $\neg F = \bigwedge_{q \in F} \neg q$ on $TS \otimes \mathcal{A}$.

if $TS \otimes \mathcal{A} \models P_{inv(\mathcal{A})}$ **then**
  return true
else
  Determine an initial path fragment $\langle s_0, q_1 \rangle \ldots \langle s_n, q_{n+1} \rangle$ of $TS \otimes \mathcal{A}$ with $q_{n+1} \in F$
  return (false, $s_0\, s_1 \ldots s_n$)
fi

The time and space complexity is in O(|TS|·|A|).
|TS| and |A| denote the number of states and of transitions in TS and A, respectively.

MODEL CHECKING *LIVENESS* PROPERTIES

Automata on Infinite words

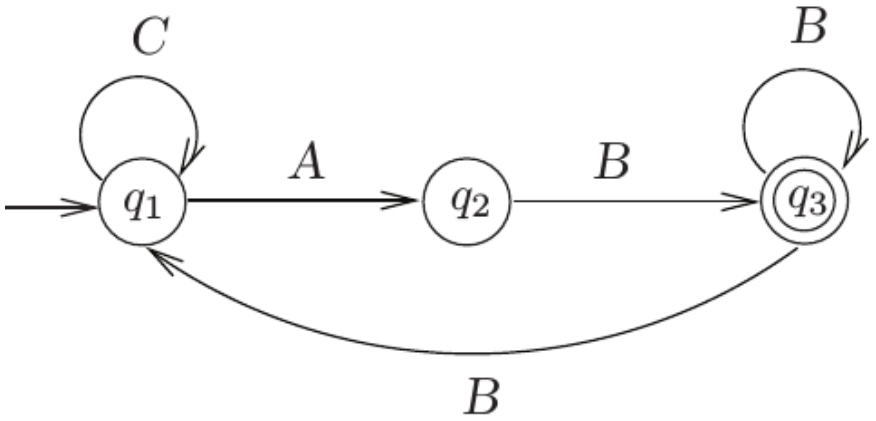
- By definition, a liveness property cannot be verified on a finite prefix of a run.
- Therefore, we need a concept of regular language over **infinite words**
  - The rough idea is to consider variants of NFAs, called nondeterministic Büchi automata (NBAs), which serve as acceptors for languages of infinite words.
  - If we are given a NBA A that specifies the "bad traces" (i.e., that accepts the complement of the LT property P to be verified), then a graph analysis in the product of the given transition system TS and the automaton A suffices to either establish or disprove TS |= P .

ω-regular languages

- Infinite words over the alphabet Σ are infinite sequences $A_0\ A_1\ A_2\ \ldots$ of symbols $A_i \in \Sigma$. Σ^ω denotes the set of all infinite words over Σ.
- Any subset of Σ^ω is called a language of infinite words, sometimes also called an ω-language.
- The name "language" will be used for any subset of $\Sigma^* \cup \Sigma^\omega$.
  - It is possible to define a concept of ω-regular language by using a generalized version of regular expression called of ω-regular expressions.
  - we prefer to define ω-regular languages using automata, but we may use ω-regular expressions to better understand an ω-regular language recognized by a BA.

Nondeterministic Büchi Automaton (NBA)

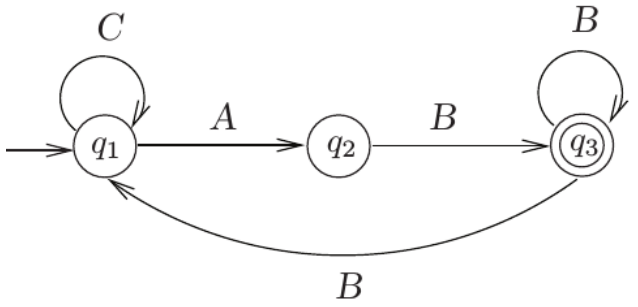
- They are formally «identical» to NFA, but they are defined to accept ω-words instead of finite ones.
- A nondeterministic Büchi automaton (NBA) A is a tuple $A = (Q, \Sigma, \delta, Q_0, F)$ where
  - Q is a finite set of states, Σ is an alphabet,
  - $\delta : Q \times \Sigma \rightarrow 2^Q$ is a transition function,
  - $Q_0 \subseteq Q$ is a set of initial states, and $F \subseteq Q$ is a set of final states, called the *acceptance set*.



Acceptance for NBA

A run for $\sigma = A_0 A_1 A_2 \ldots \in \Sigma^\omega$ denotes an infinite sequence $q_0\, q_1\, q_2 \ldots$ of states in $\mathcal{A}$ such that $q_0 \in Q_0$ and $q_i \xrightarrow{A_i} q_{i+1}$ for $i \geqslant 0$. Run $q_0\, q_1\, q_2 \ldots$ is *accepting* if $q_i \in F$ for infinitely many indices $i \in \mathbb{N}$. The *accepted language* of $\mathcal{A}$ is

$$\mathcal{L}_\omega(\mathcal{A}) \;=\; \{\, \sigma \in \Sigma^\omega \mid \text{there exists an accepting run for } \sigma \text{ in } \mathcal{A} \,\}.$$



The runs that go infinitely often through the accept state q_3 are accepting. For instance, $q_1\, q_2\, q_3^\omega$ and $(q_1\, q_1\, q_2\, q_3)^\omega$ are accepting runs. q_1^ω is not an accepting run as it never visits the accept state q_3, while runs of the form $(q_1\, q_2\, q_3)^n\, q_1^\omega$ are not accepting as they visit the accept state q_3 only finitely many times. The language accepted by this NBA is given by the ω-regular expression:

$$C^* AB \left(B^+ \;+\; BC^* AB\right)^\omega$$

«Infinitely Often Green»

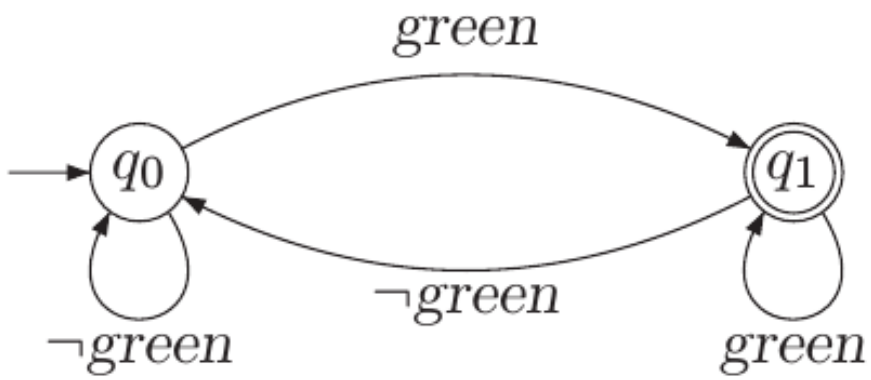


Figure 4.8: An NBA accepting "infinitely often *green*".

Also safety properties can be described by BA

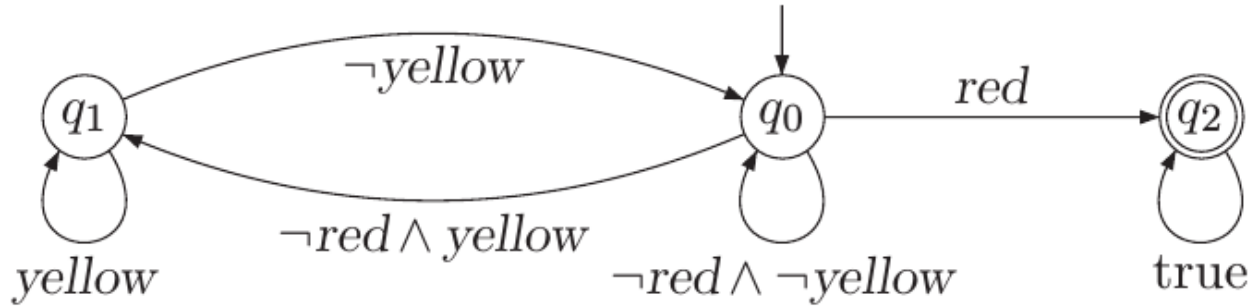
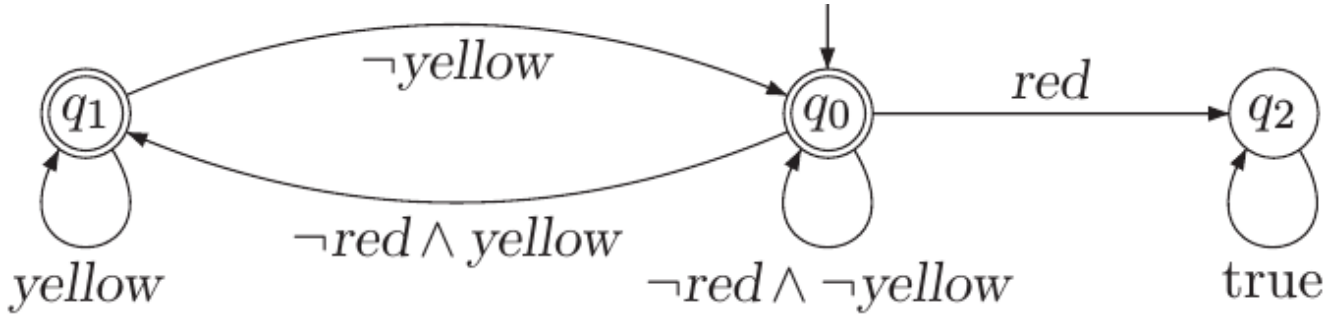


Figure 4.10: An NFA for the set of all bad prefixes of P_{safe}.

NFA accepts the negation of the property: if reaching the bad prefix "red not preceded by yellow" it goes to a final state.



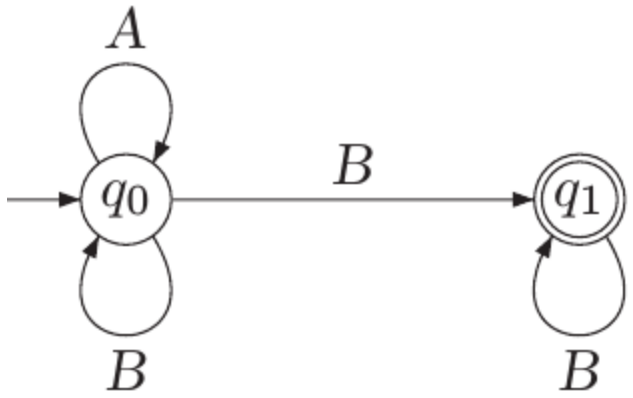
An NBA for the LT property "red should be preceded by yellow".

NBA accepst the property: if reaching the bad prefix "red not preceded by yellow", it stays i.o. in a non-final state.

Deterministic BA less powerful than NBA

Theorem 4.50. ***NBAs are More Powerful than DBAs***

There does not exist a DBA $\mathcal{A}$ such that $\mathcal{L}_\omega(\mathcal{A}) = \mathcal{L}_\omega((A+B)^*B^\omega)$.



- The idea of this NBA is that given an input word $\sigma = wB^\omega$ where $w \in \{A, B\}^*$ the automaton may stay in q_0 and guess nondeterministically when the suffix consisting of B's starts and then moves to the accept state q_1.
- This behavior, however, cannot be simulated by a DBA

Another liveness property requiring a NBA

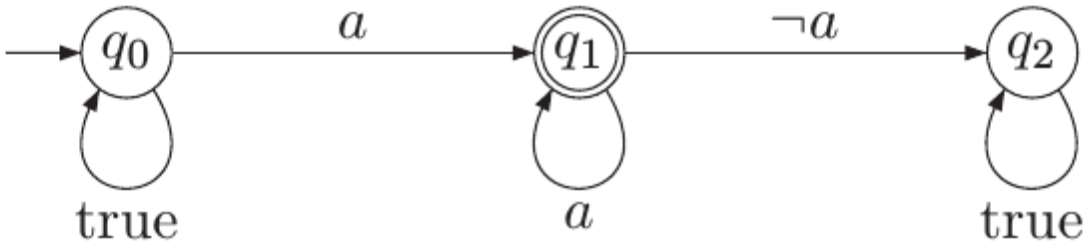


Figure 4.18: An NBA accepting "eventually forever a".

NB: There are several variants of ω-automata that are equally expressive as NBA, although they use more general acceptance conditions. For some of these ω-automata types, the deterministic version has the full power of ω-regular languages.
(Generalized BA, Rabin, Street, Mueller, etc.)

Model checking ω-regular properties

- A finite transition system TS without terminal states and an ω-regular property P. The aim is to check algorithmically whether TS |= P .
- As in the case of regular safety properties, TS |= P iff, given NBA recognizing the complement of P:

$$Traces(TS) \cap \mathcal{L}_\omega(\mathcal{A}) \neq \varnothing.$$

1) Build the product TS $\otimes$ A which combines paths in TS with the runs in A.
2) Perform a graph analysis on TS $\otimes$ A to check whether there is a path that visits a final state of A infinitely often
3) If the path exists, then it is a counterexample
4) Else all runs for the traces in TS are nonaccepting, and hence $Traces(TS) \cap \mathcal{L}_\omega(\mathcal{A}) = \varnothing$

Very rough idea of the algorithm

- Checking an ω-regular property P on a finite transition system TS can be reduced to checking the persistence property "eventually forever no accept state" in the product of TS and an NBA for the *undesired* behaviors (i.e., the *complement property of* P).
- Using a DFS, algorithm searches for a cycle in the product automaton, reachable from the initial state, and including a final state
- With some refined techniques, this algorithm may be made linear in the size of the product graph (with a nested depth-first search)—number of states + number of transitions.
- Pay attention: need to define NBA recognizing the complement of the property P. Complementing a NBA recognizing P is in practice unfeasible (between single and double exponential, more precisely $(0.76n)^n$ time)

MODEL CHECKING TEMPORAL LOGIC

Expressing properties on Transition Systems without automata

- Properties to be checked for transition systems can be expressed in many ways, but typically through *temporal logic*
- Many kinds of temporal logics
  - discrete-time vs. continuous-time
  - metric vs. non-metric
  - linear vs. branching
  - ...
- Historically, the 2 temporal logics most used in system verification approaches are
  - LTL (Linear Temporal Logic)
  - CTL (Computation Tree Logic)
- Here, we focus on LTL. We'll see CTL later.

LTL: Propositional Temporal Logic

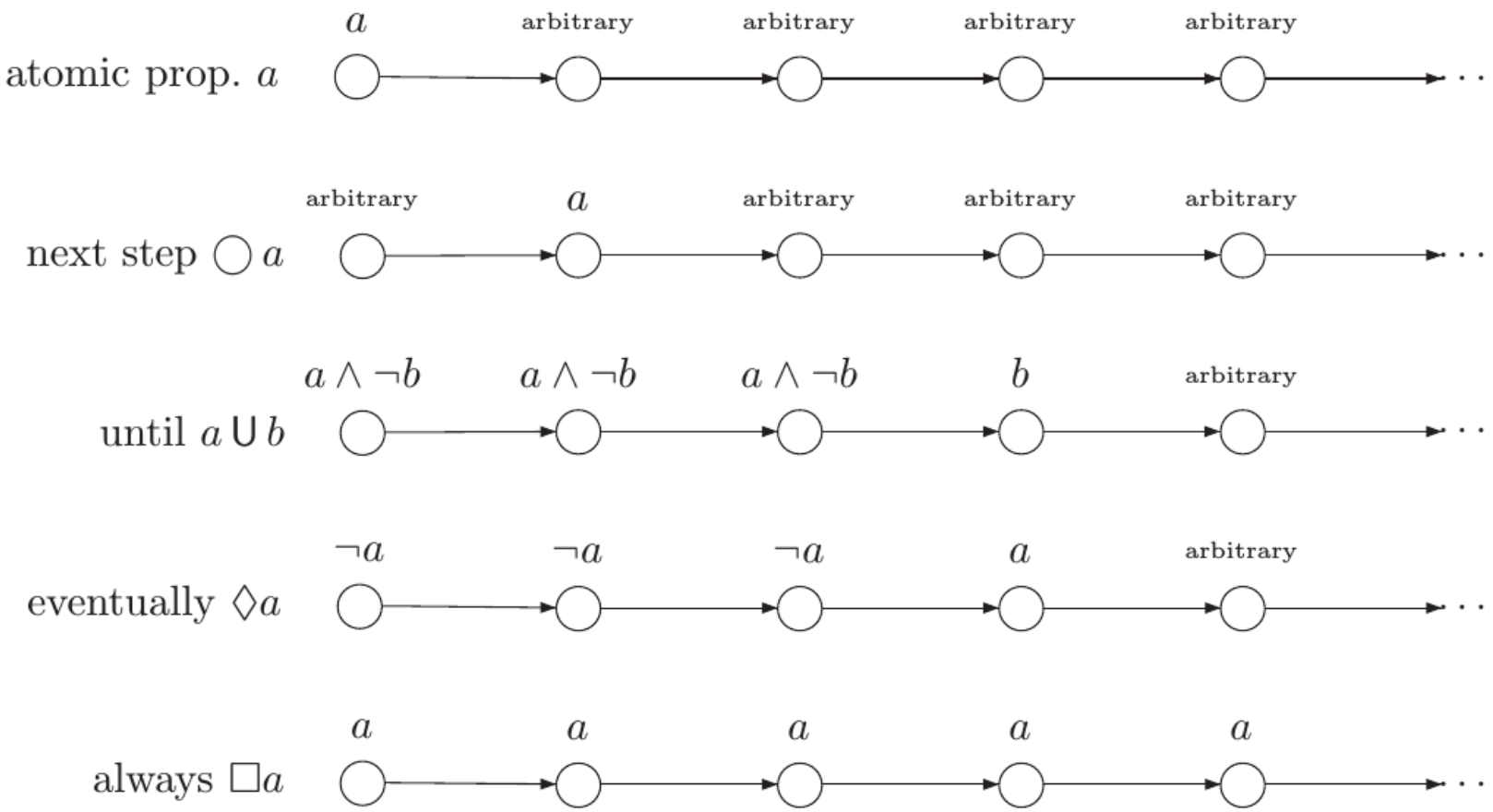
- Syntax of LTL, with a in AP

$$\varphi ::= \text{true} \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc \varphi \mid \varphi_1 \,\mathsf{U}\, \varphi_2$$

- The until operator allows to derive the temporal modalities “eventually” (sometimes in the future) and “always” (from now on forever) as follows:

$$\Diamond\varphi \stackrel{\text{def}}{=} \text{true}\,\mathsf{U}\,\varphi \qquad \Box\varphi \stackrel{\text{def}}{=} \neg\Diamond\neg\varphi$$

Intuitive semantics



Examples

Invariant: always at least one of the two processes is not in its critical section

$$\Box(\neg\, crit_1 \ \vee \ \neg\, crit_2).$$

Liveness; each process Pi is infinitely often in its critical section

$$(\Box\Diamond crit_1) \ \wedge \ (\Box\Diamond crit_2).$$

Progress: "every request will eventually lead to a response"

$$\Box(request \rightarrow \Diamond response).$$

More examples

“once red, the light cannot become green immediately”

$$\Box(red \rightarrow \neg \bigcirc\ green).$$

Once red, light always becomes green eventually after being yellow for some time”

$$\Box(red \rightarrow \bigcirc (red\, \mathsf{U}\, (yellow\ \wedge \bigcirc\ (yellow\, \mathsf{U}\, green)))).$$

Semantics of LTL over infinite words on 2^{AP}

$$
\begin{array}{lcll}
\sigma & \models & \text{true} & \\
\sigma & \models & a & \text{iff} \quad a \in A_0 \quad (\text{i.e., } A_0 \models a) \\
\sigma & \models & \varphi_1 \wedge \varphi_2 & \text{iff} \quad \sigma \models \varphi_1 \text{ and } \sigma \models \varphi_2 \\
\sigma & \models & \neg\, \varphi & \text{iff} \quad \sigma \not\models \varphi \\
\sigma & \models & \bigcirc \varphi & \text{iff} \quad \sigma[1\ldots] = A_1 A_2 A_3 \ldots \models \varphi \\
\sigma & \models & \varphi_1 \,\mathsf{U}\, \varphi_2 & \text{iff} \quad \exists j \geqslant 0.\ \sigma[j\ldots] \models \varphi_2 \ \text{ and } \ \sigma[i\ldots] \models \varphi_1, \ \text{for all } 0 \leqslant i < j
\end{array}
$$

Figure 5.2: LTL semantics (satisfaction relation $\models$) for infinite words over 2^{AP}.

Here, for $\sigma = A_0 A_1 A_2 \ldots \in (2^{AP})^\omega$, $\sigma[j\ldots] = A_j A_{j+1} A_{j+2} \ldots$ is the suffix of σ starting in the $(j+1)$st symbol A_j.

$$
\begin{array}{lcll}
\sigma & \models & \Diamond \varphi & \text{iff} \quad \exists j \geqslant 0.\ \sigma[j\ldots] \models \varphi \\
\sigma & \models & \Box \varphi & \text{iff} \quad \forall j \geqslant 0.\ \sigma[j\ldots] \models \varphi.
\end{array}
$$

«Timed» properties (over discrete time)

- Operator $\bigcirc\varphi$ can be given a timed intepretation for synchron. systems: "at the next time instant φ holds".

$$\bigcirc^k \varphi \stackrel{\text{def}}{=} \underbrace{\bigcirc \bigcirc \ldots \bigcirc}_{k\text{-times}} \varphi \qquad \text{“}\varphi \text{ holds after (exactly) } k \text{ time instants”.}$$

“φ will hold within at most k time instants”

$$\Diamond^{\leqslant k} \varphi \;=\; \bigvee_{0 \leqslant i \leqslant k} \bigcirc^i \varphi.$$

“φ holds now and will hold during the next k instants”

$$\Box^{\leqslant k} \varphi \;=\; \neg \Diamond^{\leqslant k} \neg \varphi \;=\; \neg \bigvee_{0 \leqslant i \leqslant k} \bigcirc^i \neg \varphi.$$

Past operators

- Addition of past operators (symmetrical to future ones) does not change expressive power of LTL, but it may simplify formulae
- Yesterday or "past-time and "since"

$$w, i \models \bullet\phi \quad \longleftrightarrow \quad i > 0 \text{ and } w, i-1 \models \phi$$
$$w, i \models \phi \mathcal{S} \psi \quad \longleftrightarrow \quad \exists k \geq 0 : i - k \geq 0,\ w, i-k \models \psi \text{ and } \forall 0 \leq j < k : w, i-j \models \phi.$$

Derived operators may be defined:

"Sometimes/Always in the past"

$$\blacklozenge\phi \text{ is } \text{True}\,\mathcal{S}\phi, \ \blacksquare\phi \text{ is } \neg\blacklozenge\neg\phi.$$

$$\text{'Always'' } \mathcal{A}lw, \text{ defined as } \mathcal{A}lw\ \phi := \Box\phi \wedge \blacksquare\phi.$$

Metric of time

- «metric» operators: they allow exact distance
- Bounded Until, Bounded Since, for $\sim \in \{\leq, =, \geq\}$

$$w, i \models \phi \mathcal{U}_{\sim t} \psi \longleftrightarrow \exists k \geq 0 : k \sim t,\ w, i+k \models \psi, \text{ and } \forall 0 \leq j < k : w, i+j \models \phi$$

$$w, i \models \phi \mathcal{S}_{\sim t} \psi \longleftrightarrow \exists k \geq 0 : k \sim t,\ i-k \geq 0,\ w, i-k \models \psi, \text{ and } \forall 0 \leq j < k :\ w, i-j \models \phi.$$

- Derived operators:

$\Diamond_{\sim c} \phi$ as $\text{True} \mathcal{U}_{\sim c} \phi$

$\Box_{\sim c} \phi$ is $\neg \Diamond_{\sim c} \neg \phi$

$\blacklozenge_{\sim c} \phi$ $\text{True} \mathcal{S}_{\sim c} \phi$,

$\blacksquare_{\sim c} \phi$ is $\neg \blacklozenge_{\sim c} \neg \phi$

Example: Timer reset lamp

$$(\mathrm{D1})\ \ L \longleftrightarrow \bullet(\neg OFF\, \mathcal{S}_{<\Delta} ON)$$

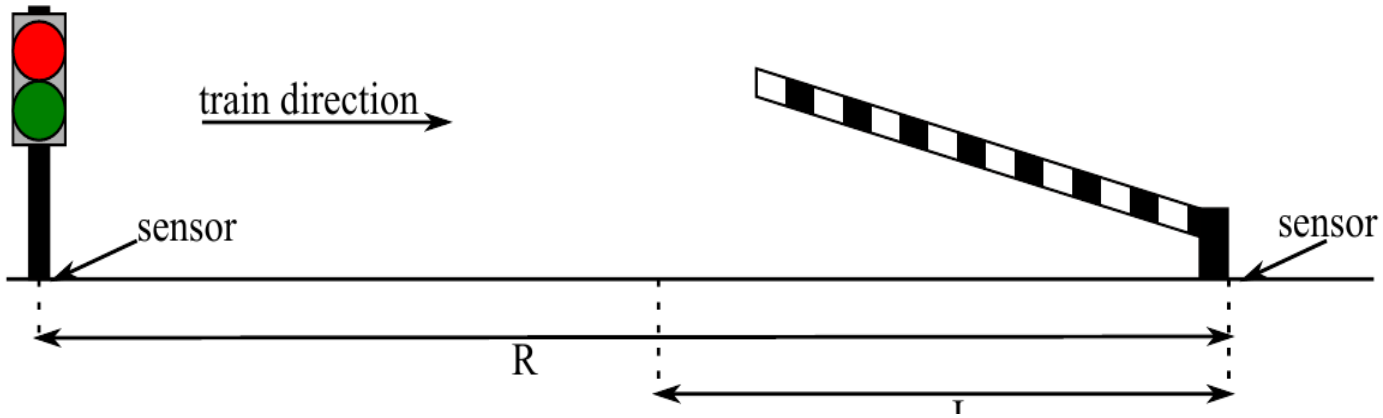
$$(D2)\ \ \neg(ON \wedge OFF)$$

$$(DM)\ \ \mathcal{A}lw(D1 \wedge D2)$$

when the ON button is pressed the lamp is lighted and it remains so, if no other event occurs, for Δ time units, after which it goes off.
Before the expiration deadline: pushing the OFF button turns off the lamp
pushing again the ON button extends the lighting.

Axiomatic approach to modeling! Using logic rather than a "machine"

KRC (with a sensor at the exit)



- A minimum time dm and a maximum time dMax to go from the beginning of R to the beginning of I
- A minimum time hm and a maximum time hMax to go from the beginning of region I to its end
- The controller operates the bar by means of *go(up)* and *go(down)* commands; the bar current position or state of motion is one of: *closed*, *open*, *movingUp* (when opening), and *movingDown* (when closing).
- It takes the bar γ time units to reach the closed (respectively, open) position starting from the open (respectively, closed) state.

Axioms

- Controller:

$$go(up) \longleftrightarrow ExitI$$

$$go(down) \longleftrightarrow \blacklozenge_{d_m - \gamma} EnterR$$

- Interlocking:

$$red \longleftrightarrow \neg green$$

$$red \longleftrightarrow \bullet(\neg(ExitI \wedge \neg EnterR)\ \mathcal{S}\ EnterR)$$

- Gate:

$$\bullet bar(closed) \wedge go(up)$$

$$\rightarrow$$

$$\Box_{<\gamma} bar(mvup) \wedge \Diamond_{=\gamma}((bar(open) \mathcal{U} go(down)) \vee \Box bar(open))$$

$$\bullet bar(open) \wedge go(down)$$

$$\rightarrow$$

$$\Box_{<\gamma} bar(mvdown) \wedge \Diamond_{=\gamma}((bar(closed) \mathcal{U} go(up)) \vee \Box bar(closed))$$

Expressing Fairness in LTL?
Action-based vs. State-based fairness

- LTL is defined over AP, not over the actions.
- So LTL fairness constraints are given over states, rather than actions
- Action-based fairness is easier and more intuitive, but LTL fairness is as expressive
  - Action-based fairness assumptions can always be “translated” into analogous LTL fairness assumptions.
  - Make a copy of each noninitial state s such that it is recorded which action was executed to enter state s. Such copy is made for every possible action α via which the state can be entered.
  - The copied state <s, α> indicates that state s has been reached by performing α as last action.

Expressing (state-based) Fairness in LTL

1. An *unconditional LTL fairness constraint* is an LTL formula of the form

$$ufair = \Box\Diamond\Psi.$$

2. A *strong LTL fairness condition* is an LTL formula of the form

$$sfair = \Box\Diamond\Phi \longrightarrow \Box\Diamond\Psi.$$

3. A *weak LTL fairness constraint* is an LTL formula of the form

$$wfair = \Diamond\Box\Phi \longrightarrow \Box\Diamond\Psi.$$

Using fairness assumptions

$$fair \;=\; ufair \;\wedge\; sfair \;\wedge\; wfair\,.$$

$$FairPaths(s) \;=\; \{\, \pi \in Paths(s) \mid \pi \models fair \,\},$$

$$s \models_{fair} \phi \quad \text{iff} \quad \forall \pi \in FairPaths(s).\, \pi \models \phi \quad \text{and}$$

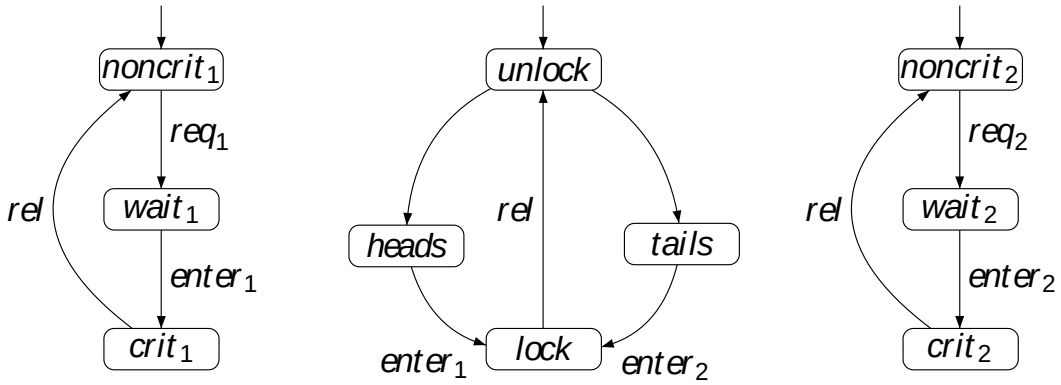
$$TS \models_{fair} \phi \quad \text{iff} \quad \forall s_0 \in I\,.\, s_0 \models_{fair} \phi.$$

Theorem 5.30. Reduction of $\models_{fair}$ to $\models$

For transition system TS without terminal states, LTL formula ϕ, and LTL fairness assumption fair:

$$TS \models_{fair} \phi \qquad \textit{if and only if} \qquad TS \models (fair \rightarrow \phi).$$

Example: randomized mutual exclusion



$$TS_1 \parallel Arbiter \parallel TS_2 \not\models \Box\Diamond\, crit_1.$$

The property "process TS1 is in its critical section infinitely often" cannot be established, since, for instance, the underlying transition system representation does not exclude an execution in which only the second process may perform an action while TS1 is entirely ignored.

If a coin is assumed to be fair enough such that both events "heads" and "tails" occur with positive probability, the unrealistic case of one of the two alternatives never happening can be ignored by means of the unconditional LTL fairness assumption:

$$fair \;=\; \Box\Diamond\, heads \wedge \Box\Diamond\, tails.$$

It is not difficult to check that now:

$$TS_1 \parallel Arbiter \parallel TS_2 \models_{fair} \Box\Diamond\, crit_1 \wedge \Box\Diamond\, crit_2.$$

LTL MODEL CHECKING

Positive Normal Form

- PNF is a canonical form such that **negations only occur adjacent to atomic propositions**.
- ¬a ∧ ((¬b ∧ c) ∨ ¬a) is in PNF, while ¬(a ∧ ¬b) is not.
- The well-known disjunctive and conjunctive normal forms are special cases of the PNF for propositional logic.
- Every LTL formula can be transformed into canonical normal form, but this requires new “dual” operators

Dual Operators

- OR and AND are dual, by De Morgan's Law
- Next-step is dual of itself: ¬ ○ φ ≡ ○ ¬φ
- Until: Dual is «Weak Until» W, defined as

$$\varphi \,\mathsf{W}\, \psi \;\stackrel{\text{def}}{=}\; (\varphi \,\mathsf{U}\, \psi) \;\vee\; \Box\varphi.$$

$$\begin{aligned} \neg(\varphi \,\mathsf{U}\, \psi) &\equiv (\varphi \wedge \neg\psi) \,\mathsf{W}\, (\neg\varphi \wedge \neg\psi) \\ \neg(\varphi \,\mathsf{W}\, \psi) &\equiv (\varphi \wedge \neg\psi) \,\mathsf{U}\, (\neg\varphi \wedge \neg\psi) \end{aligned}$$

PNF syntax

- Positive Normal Form for LTL (Weak-Until PNF)

$$\varphi \ ::= \ \text{true} \ \Big| \ \text{false} \ \Big| \ a \ \Big| \ \neg a \ \Big| \ \varphi_1 \wedge \varphi_2 \ \Big| \ \varphi_1 \vee \varphi_2 \ \Big| \ \bigcirc \varphi \ \Big| \ \varphi_1 \, \mathsf{U} \, \varphi_2 \ \Big| \ \varphi_1 \, \mathsf{W} \, \varphi_2 .$$

- Theorem: For each LTL formula there exists an equivalent LTL formula in weak-until PNF, which may be **exponential in the size of the original formula**.
- To avoid exponential blow up a different form is introduced

Release operator

- Release R defined as: $\varphi \,\mathsf{R}\, \psi \quad \stackrel{\text{def}}{=} \quad \neg(\neg\varphi \,\mathsf{U}\, \neg\psi).$
- Semantics: $\sigma \models \varphi \,\mathsf{R}\, \psi$

or

$$\forall j \geqslant 0.\ \sigma[j..] \models \psi$$
$$\exists i \geqslant 0.\ (\sigma[i..] \models \varphi) \wedge \forall k \leqslant i.\ \sigma[k..] \models \psi)$$

- Intuitive interpretation:
  - Formula *φ R ψ* holds for a word if ψ always holds, a requirement that is released as soon as φ becomes true.
  - For instance, the formula *false R φ* is valid if φ always holds, since the release condition (false) is a contradiction.

Release PNF

$$\varphi \;::=\; \text{true} \;\big|\; \text{false} \;\big|\; a \;\big|\; \neg a \;\big|\; \varphi_1 \wedge \varphi_2 \;\big|\; \varphi_1 \vee \varphi_2 \;\big|\; \bigcirc\, \varphi \;\big|\; \varphi_1 \,\mathsf{U}\, \varphi_2 \;\big|\; \varphi_1 \,\mathsf{R}\, \varphi_2.$$

- Rewriting rules that only increase the formula by a constant

$$\begin{array}{lcl}
\neg\text{true} & \rightsquigarrow & \text{false} \\
\neg\neg\varphi & \rightsquigarrow & \varphi \\
\neg(\varphi \wedge \psi) & \rightsquigarrow & \neg\varphi \vee \neg\psi \\
\neg \bigcirc\, \varphi & \rightsquigarrow & \bigcirc\, \neg\varphi \\
\neg(\varphi \,\mathsf{U}\, \psi) & \rightsquigarrow & \neg\varphi \,\mathsf{R}\, \neg\psi
\end{array}$$

For any LTL formula φ there exists an equivalent LTL formula φ' in release PNF with $|\varphi'| = \mathcal{O}(|\varphi|)$.

Automata-based LTL Model Checking

- Crucial observation: each LTL formula φ can be represented by a nondeterministic Büchi automaton

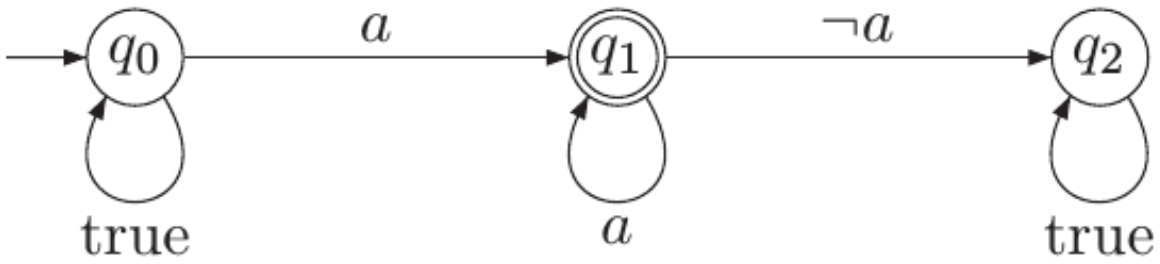


Figure 5.19: NBA for $\Diamond\Box a$.

Automata-based LTL Model Checking

- Let Words(φ) be the set of ω–words satisfying an LTL formula φ (the ω–language of φ).

$$
\begin{aligned}
TS \models \varphi \quad & \text{iff} \quad Traces(TS) \subseteq Words(\varphi) \\
& \text{iff} \quad Traces(TS) \cap ((2^{AP})^{\omega} \setminus Words(\varphi)) = \varnothing \\
& \text{iff} \quad Traces(TS) \cap Words(\neg\varphi) = \varnothing.
\end{aligned}
$$

Hence, for NBA $\mathcal{A}$ with $\mathcal{L}_{\omega}(\mathcal{A}) = Words(\neg\varphi)$ we have

$$TS \models \varphi \quad \text{if and only if} \quad Traces(TS) \cap \mathcal{L}_{\omega}(\mathcal{A}) = \varnothing.$$

NB: Instead of building a NBA equivalent to the formula and then complementing the automaton, it is much more efficient to complement the formula and then building the equivalent NBA.

From LTL to GBA

- Construction builds first a so-called Generalized Buchi Automata (GBA) and then converts GBA to NBA
- GBA have a (possibly empty) set $\mathcal{F} \subseteq 2^Q$ of «acceptance sets»
  - The accepted languages consists of all ω–words that have:
  - at least an infinite run $\mathbf{q_0\ q_1\ q_2\ \ldots}$ such that: **for each acceptance set $F \in \mathcal{F}$ there are infinitely many indices i with $q_i \in F$.**
- Each element of set $\mathcal{F}$ is defined to represent «eventuality» for every Until operator occurring in the formula

For any LTL formula φ (over AP) there exists a GNBA $\mathcal{G}_\varphi$ over the alphabet 2^{AP}

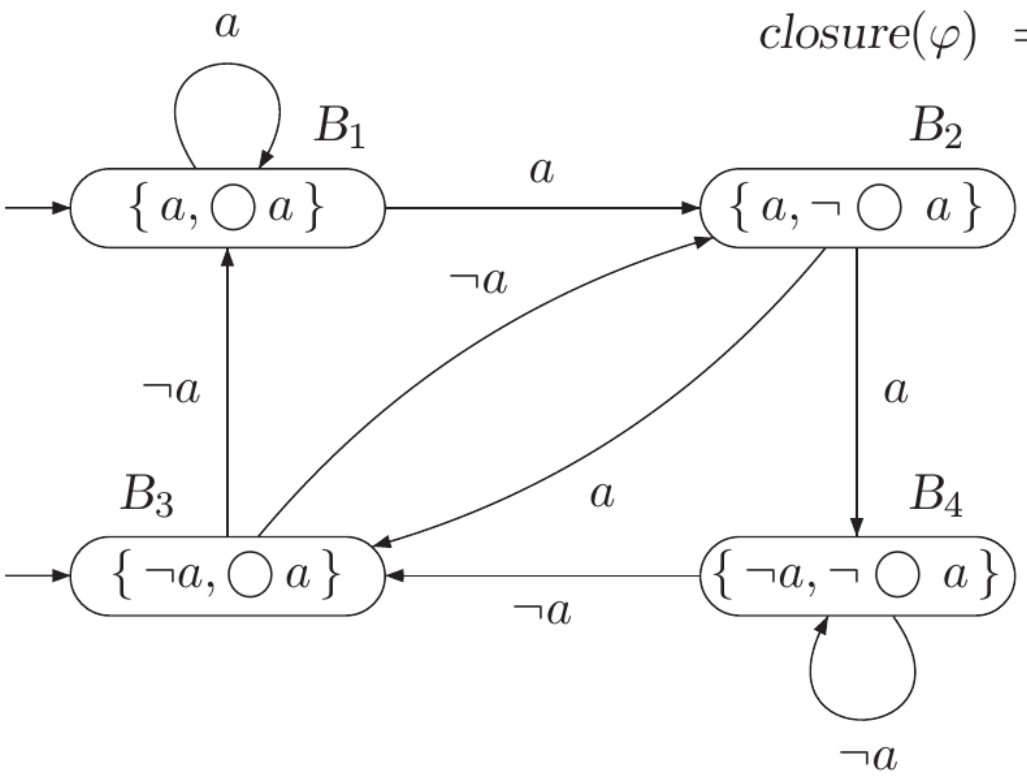
(a) $Words(\varphi) = \mathcal{L}_\omega(\mathcal{G}_\varphi)$.

(b) $\mathcal{G}_\varphi$ *can be constructed in time and space* $2^{\mathcal{O}(|\varphi|)}$.

(c) The number of accepting sets of $\mathcal{G}_\varphi$ *is bounded above by* $\mathcal{O}(|\varphi|)$.

Example: GBA/NBA for ○a

$$closure(\varphi) = \{ a, \bigcirc a, \neg a, \neg \bigcirc a \}.$$



The initial states of $\mathcal{G}_\varphi$ are the elementary sets $B \in Q$ with $\varphi = \bigcirc a \in B$.

States keep track of truth of every subformula and their negation. Ex. in B1 we must read *a* only, since *a* is in B1; in B3 only $\neg a$ can be read (since *a* is not in B3);
This GBA has $\mathcal{F}=\emptyset$ (there is no Until), and it is equivalent to the NBA where all states are final.

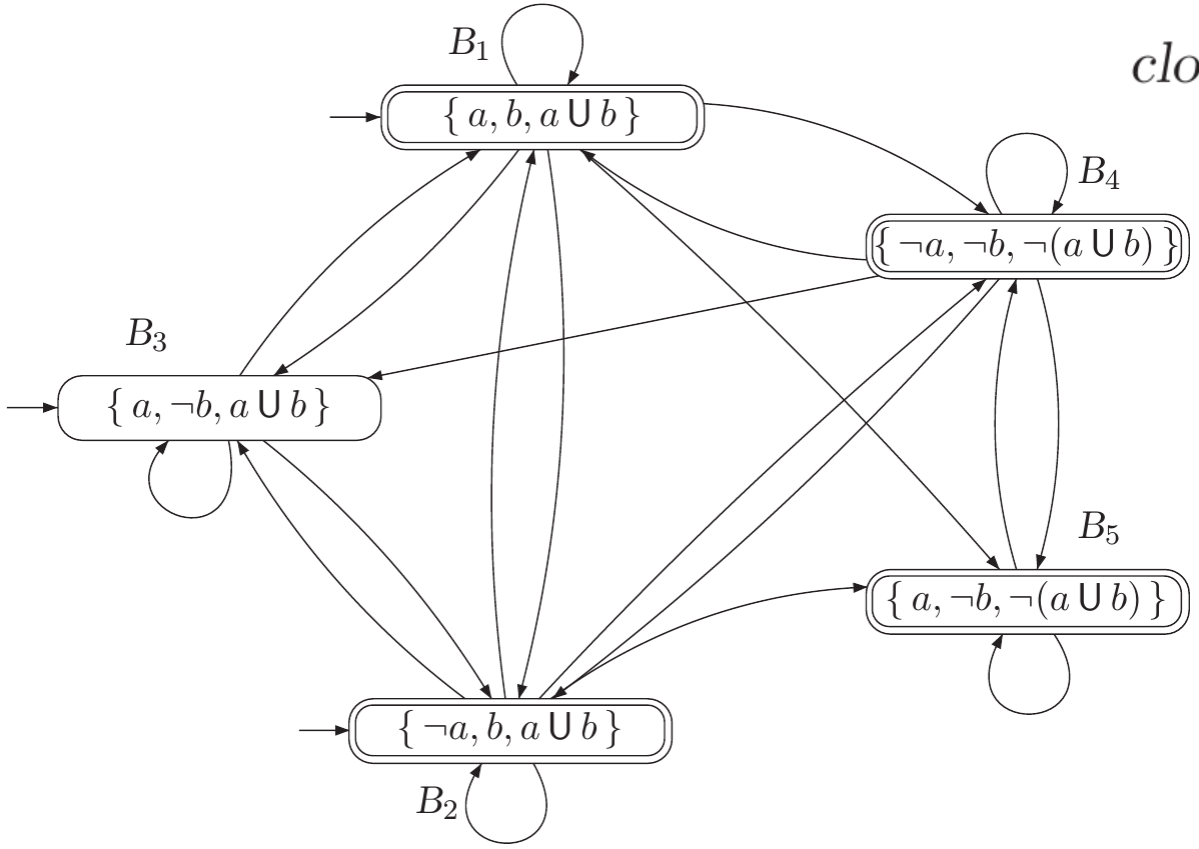
Until operator and «eventuality»

- To model the semantics of U , an acceptance set Fψ is introduced for every subformula ψ = φ1 U φ2 of φ.
- The underlying idea is to ensure the semantics of Until is verified; in every run $B_0B_1\,B_2$ … for which $\psi \in B_0$, we have:
  $\varphi_2 \in B_j$ for some j >= 0 and $\varphi_1 \in B_i$ for all i < j.
- The requirement that a word σ satisfies $\varphi_1 \,\mathsf{U}\, \varphi_2$ only if φ_2 *will eventually become true* is ensured by the accepting set F_ψ.

$$F_{\varphi_1 \,\mathsf{U}\, \varphi_2} = \{ B \in Q \mid \varphi_1 \,\mathsf{U}\, \varphi_2 \notin B \text{ or } \varphi_2 \in B \}.$$

$$\mathcal{F} = \{ F_{\varphi_1 \,\mathsf{U}\, \varphi_2} \mid \varphi_1 \,\mathsf{U}\, \varphi_2 \in closure(\varphi) \}$$

Example: GBA for φ= aUb



$$closure(\varphi) = \{ a, b, \neg a, \neg b, a \mathsf{U} b, \neg(a \mathsf{U} b) \}.$$

$$Q_0 = \{ B_1, B_2, B_3 \}$$

$$\mathcal{F} = \{ F_\varphi \}$$

$$F_\varphi = \{ B \in Q \mid \varphi \notin B \vee b \in B \} = \{ B_1, B_2, B_4, B_5 \}.$$

Since the accepting set is a singleton set, the GNBA $\mathcal{G}_\varphi$ can be understood as an NBA with the accepting set F_φ.

NBA vs LTL

Theorem 5.41. Constructing an NBA for an LTL Formula

For any LTL formula φ (over AP) there exists an NBA $\mathcal{A}_\varphi$ with $Words(\varphi) = \mathcal{L}_\omega(\mathcal{A}_\varphi)$ which can be constructed in time and space $2^{\mathcal{O}(|\varphi|)}$.

- Equivalent NBA has exponential number of states!

- But NBA are more expressive than LTL
- For instance:
  - «proposition a holds in every ***even*** position»
  - cannot be expressed by a LTL formula on alphabet $2^{\{a\}}$
  - NB: choice of the alphabet is crucial: NBA are more expressive *over the same alphabet.*

Theorem 5.42. Lower Bound for NBA from LTL Formulae

There exists a family of LTL formulae φ_n with $|\varphi_n| = \mathcal{O}(poly(n))$ such that every NBA for φ_n has at least 2^n states.

$$\mathcal{L}_n = \{ A_1 \dots A_n \, A_1 \dots A_n \, \sigma \mid A_i \subseteq AP \wedge \sigma \in (2^{AP})^\omega \}, \qquad \text{for } n \geqslant 0.$$

$$\varphi_n = \bigwedge_{a \in AP} \bigwedge_{0 \leqslant i < n} (\bigcirc^i a \longleftrightarrow \bigcirc^{n+i} a). \qquad |\varphi_n| \in \mathcal{O}(|AP| \cdot n)$$

any NBA $\mathcal{A}$ with $\mathcal{L}_\omega(\mathcal{A}) = \mathcal{L}_n$ has at least 2^n states.

Proof: Words $A_1 \dots A_n \, A_1 \dots A_n \, \emptyset\emptyset\emptyset\dots$ are in $\mathcal{L}_n$, hence NBA $\mathcal{A}$ contains for every word $A_1 \dots A_n$ of length n, a state $q(A_1 \dots A_n)$, which can be reached from an initial state by consuming the prefix $A_1 \dots A_n$.

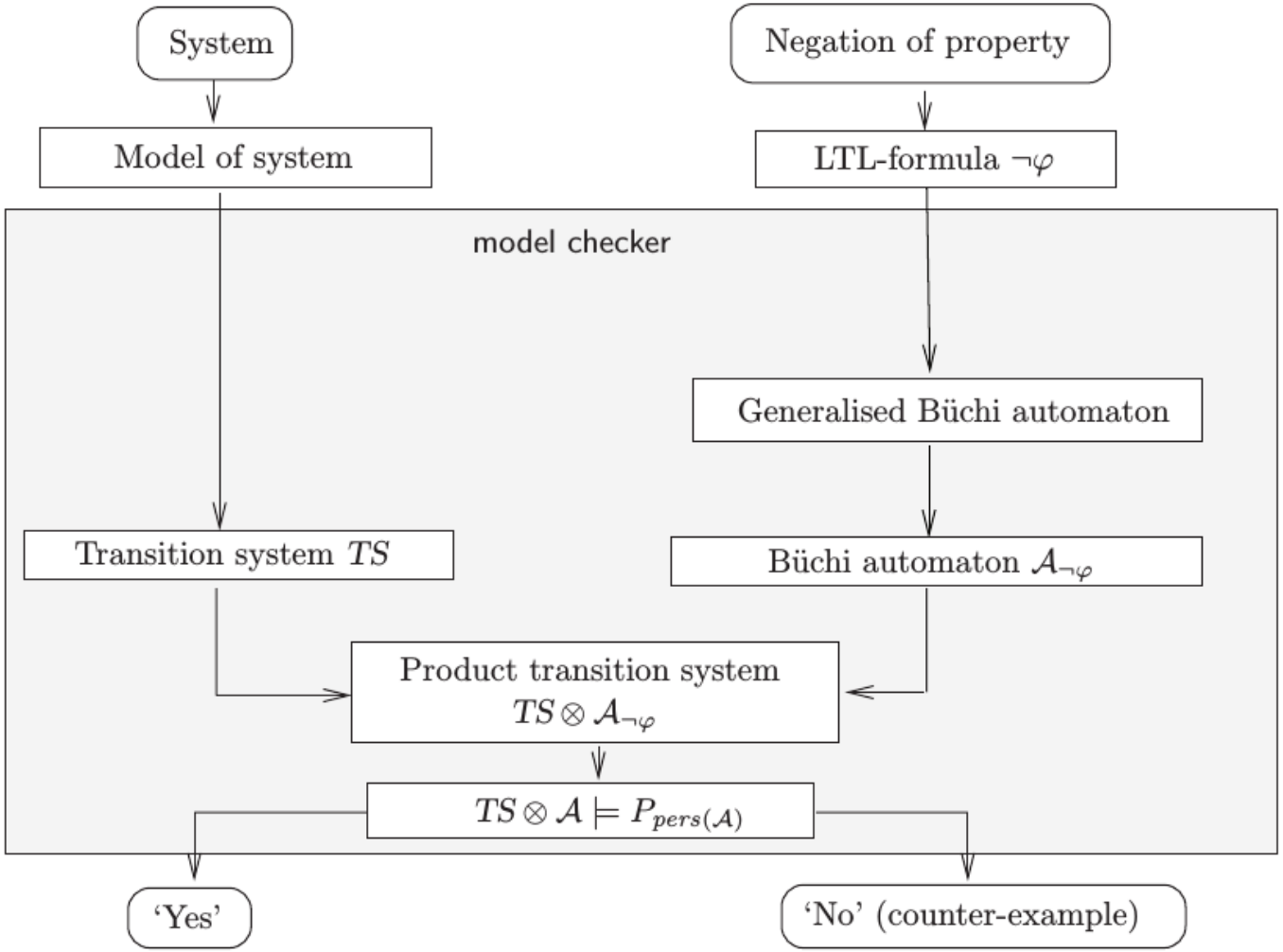
From $q(A_1 \dots A_n)$ it is possible to visit an accept state ∞-often by accepting suffix $A_1 \dots A_n \emptyset\emptyset\emptyset\dots$

But $A_1 \dots A_n A'_1 \dots A'_n \; \emptyset \; \emptyset \; \emptyset \; \dots$ is not in $\mathcal{L}_n$ for $A_1 \dots A_n \neq A'_1 \dots A'_n$
Therefore, the states $q(A_1 \dots A_n)$ are all pairwise different.

As there are $|2^{AP}|$ possible combinations for each A_i, $\mathcal{A}$ has at least $(|2^{AP}|)^n >= 2^n$ states.

Overview

Time and Space complexity: PSPACE-complete $\mathcal{O}(\,|TS| \cdot 2^{|\varphi|}\,)$.
(but in practice good performance)



On-the fly-construction

- An interesting aspect of the LTL model-checking algorithm is that it can be executed on-the-fly, i.e., while constructing the NBA $A_{\neg\varphi}$. This may avoid the need for constructing the entire automaton $A_{\neg\varphi}$.
- This is exploited for instance by the SPIN model checker.

Example

```
int x = 100;
active proctype A() {
        do  :: x %2 -> x = 3*x+1
        od
  }
 active proctype B(){
        do  :: !(x%2) -> x = x/2
        od
 }
```

Property to be verified: **GF (x = 1)** G is Globally, F (future) is Eventually)

- NB: x=1 is not an atomic proposition, but just add: **#define q (x == 1)**
- Verify **GF q**
- In practice, SPIN defines on-the-fly the intersection of the Promela code with the NBA for ¬(GF q) : if property is not verified, a counterexample is returned

Checking satisfiability of LTL formulae

Satisfiability and validity of LTL formulae can be solved via checking emptiness of NBA. The emptiness check can be determined by a nested DFS that checks the existence of a reachable cycle containing an accept state. Both problems are PSPACE-complete.

Algorithm 12 Satisfiability checking for LTL

Input: LTL formula φ over AP
Output: "yes" if φ is satisfiable. Otherwise "no".

Construct an NBA $\mathcal{A} = (Q, 2^{AP}, \delta, Q_0, F)$ with $\mathcal{L}_\omega(\mathcal{A}) = \mathit{Words}(\varphi)$

(* Check whether $\mathcal{L}_\omega(\mathcal{A}) = \varnothing$. *)

Perform a nested DFS to determine whether there exists a state $q \in F$ reachable from $q_0 \in Q_0$ and that lies on a cycle

If so, then return "yes". Otherwise, "no".