Automata-Theoretic Approach to Model Checking

(Based on [Clarke *et al.* 1999], [Manna and Pnueli 1995], and [Kesten and Pnueli 2002])

Yih-Kuen Tsay

(original created by Wen-Chin Chan)

Dept. of Information Management

National Taiwan University

Outline

- **Büchi Automata**
- Model Checking Using Automata
- Checking Emptiness
- Simple On-the-fly Translation
- Tableau Construction
- Inductive Construction

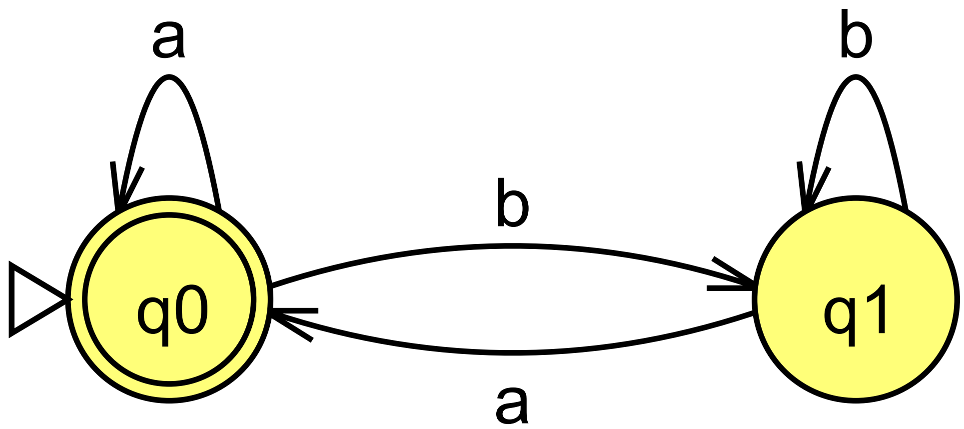
Finite Automata

- A finite automaton is a mathematical model of a device that has a constant amount of memory, independent of the size of its input.
- Formally, a **finite automaton** (FA) is a 5-tuple $(\Sigma, Q, \Delta, q_0, F)$, where
  1. Σ is a finite set of symbols (the *alphabet*),
  2. Q is a finite set of *states*,
  3. $\Delta \subseteq Q \times \Sigma \times Q$ is the *transition relation*,
  4. $q_0 \in Q$ is the *start* state (sometimes we allow multiple start states, indicated by Q_0 or Q^0), and
  5. $F \subseteq Q$ is the set of *final* (or accepting) states.

Finite Automata (cont.)

- Let $M = (\Sigma, Q, \Delta, q_0, F)$ be an FA and $w = w_1 w_2 \ldots w_n$ be a string (or word) over Σ.
- A *run* of M over w is a sequence of states $r_0, r_1, \ldots, r_n$ such that
  1. $r_0 = q_0$ and
  2. $(r_i, w_{i+1}, r_{i+1}) \in \Delta$ for $i = 0, 1, \ldots, n-1$.
- A run is *accepting* if it ends in a final state.
- We say that M *accepts* w if it has an accepting run over w.
- The *language* of M, denoted $L(M)$, is the set of all words that are accepted by M.

An Example Finite Automaton



- This FA accepts the empty string or strings over $\{a, b\}$ that end with an a.
- Using a regular expression, its language is expressed as $\varepsilon + (a + b)^* a$.

Büchi Automata

- To model non-terminating systems, we interpret finite automata over *infinite* words.
- The simplest finite automata over infinite words are Büchi automata (BA).
- A BA has the same structure as an FA and is also given by a 5-tuple $(\Sigma, Q, \Delta, q_0, F)$.
- Runs of a BA over infinite words are defined similarly.
- An infinite word $w \in \Sigma^\omega$ is *accepted* by a BA B if there exists a run ρ of B over w satisfying the condition:

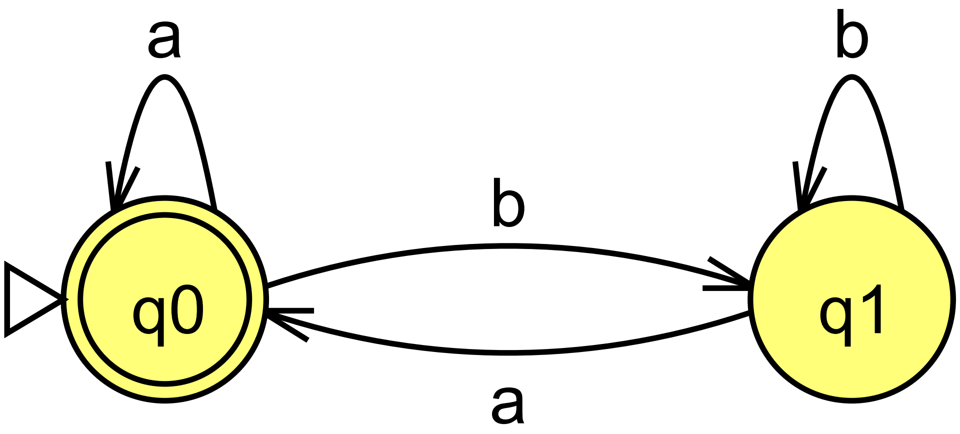
$$\mathit{inf}(\rho) \cap F \neq \emptyset,$$

where $\mathit{inf}(\rho)$ denotes the set of states occurring infinitely many times in ρ.

Büchi Automata (cont.)

- Büchi automata are a member of a larger family of the so-called ω-automata, which all have the same structure as finite automata but with different forms of acceptance conditions for the input words.
- Unlike FAs, non-determinism adds expressive power to BAs.
- Every LTL formula has an equivalent BA (but not vice versa), when infinite words are seen as models for temporal formulae.
- BAs are expressively equivalent to QPTL, a variant of LTL with quantification over atomic propositions.

An Example Finite Automaton



- This Büchi automaton accepts infinite words over $\{a, b\}$ that have infinitely many a's.
- Using an ω-regular expression, its language is expressed as $(b^*a)^\omega$.

Outline

- Büchi Automata
- **Model Checking Using Automata**
- Checking Emptiness
- Simple On-the-fly Translation
- Tableau Construction
- Inductive Construction

Modeling Concurrent Systems

- Let AP be a set of atomic propositions.
- A Kripke structure M over AP is a four-tuple $M = (S, R, S_0, L)$:
  1. S is a finite set of states.
  2. $R \subseteq S \times S$ is a transition relation that must be total, that is, for every state $s \in S$ there is a state $s' \in S$ such that $R(s, s')$.
  3. $S_0 \subseteq S$ is the set of initial states.
  4. $L : S \to 2^{AP}$ is a function that labels each state with the set of atomic propositions true in that state.

Modeling Concurrent Systems (cont.)

- Finite automata can be used to model concurrent and interactive systems.
- One of the main advantages of using automata for model checking is that both the modeled system and the specification are represented in the same way.
- A Kripke structure directly corresponds to a Büchi automaton, where all the states are accepting.
- A Kripke structure (S, R, S_0, L) can be transformed into an automaton $A = (\Sigma, S \cup \{\iota\}, \Delta, \{\iota\}, S \cup \{\iota\})$ with $\Sigma = 2^{AP}$ where
    - $(s, \alpha, s') \in \Delta$ for $s, s' \in S$ iff $(s, s') \in R$ and $\alpha = L(s')$ and
    - $(\iota, \alpha, s) \in \Delta$ iff $s \in S_0$ and $\alpha = L(s)$.

Model Checking Using Automata

- The given (finite-state) system is modeled as a Büchi automaton A.
- A desired property is given by a linear temporal formula f.
- Let B_f (resp. $B_{\neg f}$) denote a Büchi automaton equivalent to f (resp. $\neg f$).
- The model checking problem $A \models f$ is equivalent to asking whether

$$L(A) \subseteq L(B_f) \; or \; L(A) \cap L(B_{\neg f}) = \emptyset.$$

- The well-used model checker SPIN, for example, adopts this automata-theoretic approach.

Intersection of Büchi Automata

- Let $B_1 = (\Sigma, Q_1, \Delta_1, Q_1^0, F_1)$ and $B_2 = (\Sigma, Q_2, \Delta_2, Q_2^0, F_2)$.
- We can build an automaton for $L(B_1) \cap L(B_2)$ as follows.
- $B_1 \cap B_2 =$ $(\Sigma, Q_1 \times Q_2 \times \{0, 1, 2\}, \Delta, Q_1^0 \times Q_2^0 \times \{0\}, Q_1 \times Q_2 \times \{2\})$.
- We have $(\langle r, q, x\rangle, a, \langle r', q', y\rangle) \in \Delta$ iff the following conditions hold:
  - $(r, a, r') \in \Delta_1$ and $(q, a, q') \in \Delta_2$.
  - The third component is affected by the accepting conditions of B_1 and B_2.
    - If $x = 0$ and $r' \in F_1$, then $y = 1$.
    - If $x = 1$ and $q' \in F_2$, then $y = 2$.
    - If $x = 2$, then $y = 0$.
    - Otherwise, $y = x$.

Intersection of Büchi Automata (cont.)

- The third component is responsible for guaranteeing that accepting states from both B_1 and B_2 appear infinitely often (need not be at the same time).
- A simpler intersection may be obtained when all of the states of one of the automata are accepting.
- Assuming all states of B_1 are accepting and that the acceptance set of B_2 is F_2, their intersection can be defined as follows:

$$B_1 \cap B_2 = (\Sigma, Q_1 \times Q_2, \Delta', Q_1^0 \times Q_2^0, Q_1 \times F_2)$$

where $(\langle r, q\rangle, a, \langle r', q'\rangle) \in \Delta'$ iff $(r, a, r') \in \Delta_1$ and $(q, a, q') \in \Delta_2$.

Generalized Büchi Automata

- A generalized Büchi automaton (GBA) has an acceptance component of the form $F = \{F_1, F_2, \cdots, F_n\} \subseteq 2^Q$.
- A run ρ of a GBA is accepting if for each $F_i \in F$, $inf(\rho) \cap F_i \neq \emptyset$.
- There is a simple translation from a GBA to a Büchi automaton.

Generalized Büchi Automata (cont.)

- Let $B = (\Sigma, Q, \Delta, Q^0, F)$, where $F = \{F_1, \cdots, F_n\}$, be a GBA.
- Construct $B' = (\Sigma, Q \times \{0, \cdots, n\}, \Delta', Q^0 \times \{0\}, Q \times \{n\})$.
- The transition relation Δ' is constructed such that $(\langle q, x\rangle, a, \langle q', y\rangle) \in \Delta'$ when $(q, a, q') \in \Delta$ and x and y are defined according to the following rules:
  - If $q' \in F_i$ and $x = i - 1$, then $y = i$.
  - If $x = n$, then $y = 0$.
  - Otherwise, $y = x$.

Outline

- Büchi Automata
- Model Checking Using Automata
- **Checking Emptiness**
- Simple On-the-fly Translation
- Tableau Construction
- Inductive Construction

Checking Emptiness

- Let ρ be an accepting run of a Büchi automaton $B = (\Sigma, Q, \Delta, Q^0, F)$.
- Then, ρ contains infinitely many accepting states from F.
- Since Q is finite, there is some suffix ρ' of ρ such that every state on it appears infinitely many times.
- Each state on ρ' is reachable from any other state on ρ'.
- Hence, the states in ρ' are included in a strongly connected component.
- This component is reachable from an initial state and contains an accepting state.

Checking Emptiness (cont.)

- Conversely, any strongly connected component that is reachable from an initial state and contains an accepting state generates an accepting run of the automaton.
- Thus, checking nonemptiness of $L(B)$ is equivalent to finding a strongly connected component that is reachable from an initial state and contains an accepting state.
- That is, the language $L(B)$ is nonempty iff there is a reachable accepting state with a cycle back to itself.

Double DFS Algorithm

```
procedure emptiness
    for all q_0 ∈ Q^0 do
        dfs1(q_0);
    terminate(True);
end procedure

procedure dfs1(q)
    local q';
    hash(q);
    for all successors q' of q do
        if q' not in the hash table then dfs1(q');
    if accept(q) then dfs2(q);
end procedure
```

Double DFS Algorithm (cont.)

```
procedure dfs2(q)
    local q';
    flag(q);
    for all successors q' of q do
        if q' on dfs1 stack then terminate(False);
        else if q' not flagged then dfs2(q');
        end if;
end procedure
```

Correctness of the Algorithm

- **Lemma 23**
  Let q be a node that does not appear on any cycle. Then the DFS algorithm will backtrack from q only after all the nodes that are reachable from q have been explored and backtracked from.

- **Theorem 7**
  The double DFS algorithm returns a counterexample for the emptiness of the checked automaton B exactly when the language $L(B)$ is not empty.

Proof of Theorem 7

- Suppose a second DFS is started from a state q and there is a path from q to some state p on the search stack of the first DFS.
- There are two cases:
  - There exists a path from q to a state on the search stack of the first DFS that contains only unflagged nodes when the second DFS is started from q.
  - On every path from q to a state on the search stack of the first DFS there exists a state r that is already flagged.
- The algorithm will find a cycle in the first case.
- We show that the second case is impossible.

Proof of Theorem 7 (cont.)

- Suppose the contrary: On every path from q to a state on the search stack of the first DFS there exists a state r that is already flagged.
- Then there is an accepting state from which a second DFS starts but fails to find a cycle even though one exists.
  - Let q be the first such state.
  - Let r be the first flagged state that is reached from q during the second DFS and is on a cycle through q.
  - Let q' be the accepting state that starts the second DFS in which r was first encountered.
- Thus, according to our assumptions, a second DFS was started from q' before a second DFS was started from q.

Proof of Theorem 7 (cont.)

- Case 1: The state q' is reachable from q.
  - There is a cycle $q' \to \cdots \to r \to \cdots \to q \to \cdots \to q'$.
  - This cycle could not have been found previously.
  - This contradicts our assumption that q is the first accepting state from which the second DFS missed a cycle.
- Case 2: The state q' is not reachable from q.
  - q' cannot appear on a cycle.
  - q is reachable from r and q'.
  - If q' does not occur on a cycle, by Lemma 23 we must have backtracked from q in the first DFS before from q'.
  - This contradicts our assumption about the order of doing the second DFS.

Outline

- Büchi Automata
- Model Checking Using Automata
- Checking Emptiness
- **Simple On-the-fly Translation**
- Tableau Construction
- Inductive Construction

PTL (LTL with Past)

- $(\sigma, i) \models \bigcirc p \iff (\sigma, i+1) \models p$.
- $(\sigma, i) \models \Box p \iff \forall k \geq i : (\sigma, k) \models p$.
- $(\sigma, i) \models \Diamond p \iff \exists k \geq i : (\sigma, k) \models p$.
- $(\sigma, i) \models p \, \mathcal{U} \, q \iff$ for some $k \geq i$, $(\sigma, k) \models q$ and $(\sigma, j) \models p$ for all j, $i \leq j \leq k$.
- $(\sigma, i) \models p \, \mathcal{W} \, q \iff$ for some $k \geq i$, $(\sigma, k) \models q$ and $(\sigma, j) \models p$ for all j, $i \leq j \leq k$, or $(\sigma, j) \models p$ for all $j \geq i$.
- $(\sigma, i) \models p \, \mathcal{R} \, q \iff$ for all $j \geq 0$, $(\sigma, i) \not\models p$ for every $i < j$ implies $(\sigma, j) \models q$.

PTL (cont.)

- $(\sigma, i) \models \circledsim p \iff (i > 0) \rightarrow ((\sigma, i-1) \models p)$.
- $(\sigma, i) \models \ominus p \iff i > 0$ and $(\sigma, i-1) \models p$.
- $(\sigma, i) \models \boxminus p \iff \forall k : 0 \leq k \leq i : (\sigma, k) \models p$.
- $(\sigma, i) \models \diamondminus p \iff \exists k : 0 \leq k \leq i : (\sigma, k) \models p$.
- $(\sigma, i) \models p \, \mathcal{S} \, q \iff$ for some $k \leq i$, $(\sigma, k) \models q$ and $(\sigma, j) \models p$ for all j, $k < j \leq i$.
- $(\sigma, i) \models p \, \mathcal{B} \, q \iff$ for some $k \leq i$, $(\sigma, k) \models q$ and $(\sigma, j) \models p$ for all j, $k < j \leq i$, or $(\sigma, j) \models p$ for all $j \leq i$.

Simple On-the-fly Translation

- This is a tableau-based algorithm for obtaining an automaton from an LTL formula.
- The algorithm is geared towards being used in model checking in an on-the-fly fashion:

    It is possible to detect that a property does not hold by only constructing part of the model and of the automaton.

- The algorithm can also be used to check the validity of a temporal logic assertion.

Preprocessing of Formulae

To apply the translation algorithm, we first put the formula φ into *negation normal form*:

- $\Diamond p = True\,\mathcal{U}\,p$
- $\Box p = False\,\mathcal{R}\,p$
- $\neg(p\,\mathcal{U}\,q) = (\neg p)\,\mathcal{R}\,(\neg q)$
- $\neg(p\,\mathcal{R}\,q) = (\neg p)\,\mathcal{U}\,(\neg q)$
- $\neg\bigcirc p = \bigcirc\neg p$

Data Structure of an Automaton Node

- *ID*: A string that identifies the node.
- *Incoming*: The incoming edges represented by the IDs of the nodes with an outgoing edge leading to the current node.
- *New*: A set of subformulae that must hold at the current state and have not yet been processed.
- *Old*: The subformulae that must hold in the node and have already been processed.
- *Next*: The subformulae that must hold in all states that are immediate successors of states satisfying the properties in *Old*.

The Algorithm

- The algorithm starts with a single node, which has a single incoming edge labeled *init* (i.e., from an initial node) and expands the nodes in an DFS manner.
- This starting node has initially one new obligation in *New*, namely φ, and *Old* and *Next* are initially empty.
- With the current node N, the algorithm checks if there are unprocessed obligations left in *New*.
- If not, the current node is fully processed and ready to be added to *Nodes*.
- If there already is a node in *Nodes* with the same obligations in both its *Old* and *Next* fields, the incoming edges of N are incorporated into those of the existing node.

The Algorithm (cont.)

- If no such node exists in *Nodes*, then the current node N is added to this list, and a new current node is formed for its successor as follows:
  1. There is initially one edge from N to the new node.
  2. *New* is set initially to the *Next* field of N.
  3. *Old* and *Next* of the new node are initially empty.
- When processing the current node, a formula η in *New* is removed from this list.
- In the case that η is a literal (a proposition or the negation of a proposition), then
  - if $\neg\eta$ is in *Old*, the current node is discarded;
  - otherwise, η is added to *Old*.

The Algorithm (cont.)

- When η is not a literal, the current node can be split into two or not split, and new formulae can be added to the fields *New* and *Next*.
- The exact actions depend on the form of η:
  - $\eta = p \wedge q$, then both p and q are added to *New*.
  - $\eta = p \vee q$, then the node is split, adding p to *New* of one copy, and q to the other.
  - $\eta = p\,\mathcal{U}\,q$ ($\cong q \vee (p \wedge \bigcirc(p\,\mathcal{R}\,q))$), then the node is split.
    For the first copy, p is added to *New* and $p\,\mathcal{U}\,q$ to *Next*.
    For the other copy, q is added to *New*.
  - $\eta = p\,\mathcal{R}\,q$ ($\cong (q \wedge p) \vee (q \wedge \bigcirc(p\,\mathcal{R}\,q))$), similar to $\mathcal{U}$.
  - $\eta = \bigcirc p$, then p is added to *Next*.

Nodes to GBA

The list of nodes in *Nodes* can now be converted into a generalized Büchi automaton $B = (\Sigma, Q, q_0, \Delta, F)$:

1. Σ consists of sets of propositions from AP.
2. The set of states Q includes the nodes in *Nodes* and the additional initial state q_0.
3. $(r, \alpha, r') \in \Delta$ iff $r \in \ Incoming(r')$ and α satisfies the conjunction of the negated and nonnegated propositions in $Old(r')$
4. q_0 is the initial state, playing the role of *init*.
5. F contains a separate set F_i of states for each subformula of the form $p\,\mathcal{U}\,q$; F_i contains all the states r such that either $q \in Old(r)$ or $p\,\mathcal{U}\,q \notin Old(r)$.

Outline

- Büchi Automata
- Model Checking Using Automata
- Checking Emptiness
- Simple On-the-fly Translation
- **Tableau Construction**
- Inductive Construction

Tableau Construction

- We next study the Tableau Construction as described in [Manna and Pnueli 1995], which handles both future and past temporal operators.
- More efficient constructions exist, but this construction is relatively easy to understand.
- A tableau is a graphical representation of all models/sequences that satisfy the given temporal logic formula.
- The construction results in essentially a GBA, but leaving propositions on the states (rather than moving them to the incoming edges of a state).
- Our presentation will be slightly different, to make the resulting GBA more apparent.

Expansion Formulae

- The requirement that a temporal formula holds at a position j of a model can often be decomposed into requirements that
  - a simpler formula holds at the same position and
  - some other formula holds either at $j+1$ or $j-1$.
- For this decomposition, we have the following expansion formulae:

$$
\begin{array}{ll}
\Box p \cong p \wedge \bigcirc \Box p & \boxminus p \cong p \wedge \tilde{\ominus} \boxminus p \\
\Diamond p \cong p \vee \bigcirc \Diamond p & \diamondminus p \cong p \vee \ominus \diamondminus p \\
p \,\mathcal{U}\, q \cong q \vee (p \wedge \bigcirc(p \,\mathcal{U}\, q)) & p \,\mathcal{S}\, q \cong q \vee (p \wedge \ominus(p \,\mathcal{S}\, q)) \\
p \,\mathcal{W}\, q \cong q \vee (p \wedge \bigcirc(p \,\mathcal{W}\, q)) & p \,\mathcal{B}\, q \cong q \vee (p \wedge \tilde{\ominus}(p \,\mathcal{B}\, q))
\end{array}
$$

Note: $p \,\mathcal{R}\, q \cong (q \wedge p) \vee (q \wedge \bigcirc(p \,\mathcal{R}\, q))$.

Closure

- We define the closure of a formula φ, denoted by Φ_φ, as the smallest set of formulae satisfying the following requirements:
  - $\varphi \in \Phi_\varphi$.
  - For every $p \in \Phi_\varphi$, if q a subformula of p then $q \in \Phi_\varphi$.
  - For every $p \in \Phi_\varphi$, $\neg p \in \Phi_\varphi$.
  - For every $\psi \in \{\Box p, \Diamond p, p\,\mathcal{U}\,q, p\,\mathcal{W}\,q\}$, if $\psi \in \Phi_\varphi$ then $\bigcirc\psi \in \Phi_\varphi$.
  - For every $\psi \in \{\diamondminus p, p\,\mathcal{S}\,q\}$, if $\psi \in \Phi_\varphi$ then $\ominus\psi \in \Phi_\varphi$.
  - For every $\psi \in \{\boxminus p, p\,\mathcal{B}\,q\}$, if $\psi \in \Phi_\varphi$ then $\widetilde{\ominus}\psi \in \Phi_\varphi$.
- So, the closure Φ_φ of a formula φ includes all formulae that are relevant to the truth of φ.

Classification of Formulae

α	$K(\alpha)$
$p \wedge q$	$p,\ q$
$\Box p$	$p,\ \bigcirc \Box p$
$\boxminus p$	$p,\ \tilde{\ominus} \boxminus p$

β	$K_1(\beta)$	$K_2(\beta)$
$p \vee q$	p	q
$\Diamond p$	p	$\bigcirc \Diamond p$
$\diamondminus p$	p	$\ominus \diamondminus p$
$p\, \mathcal{U}\, q$	q	$p,\ \bigcirc(p\, \mathcal{U}\, q)$
$p\, \mathcal{W}\, q$	q	$p,\ \bigcirc(p\, \mathcal{W}\, q)$
$p\, \mathcal{S}\, q$	q	$p,\ \ominus(p\, \mathcal{S}\, q)$
$p\, \mathcal{B}\, q$	q	$p,\ \tilde{\ominus}(p\, \mathcal{B}\, q)$

- An α-formula φ holds at position j iff all the $K(\varphi)$-formulae hold at j.
- A β-formula ψ holds at position j iff either $K_1(\psi)$ or all the $K_2(\psi)$-formulae (or both) hold at j.

Atoms

- We define an atom over φ to be a subset $A \subseteq \Phi_\varphi$ satisfying the following requirements:
  - R_{sat} : the conjunction of all state formulae in A is satisfiable.
  - $R_\neg$: for every $p \in \Phi_\varphi$, $p \in A$ iff $\neg p \notin A$.
  - R_α : for every α-formula $p \in \Phi_\varphi$, $p \in A$ iff $K(p) \subseteq A$.
  - R_β : for every β-formula $p \in \Phi_\varphi$, $p \in A$ iff either $K_1(p) \in A$ or $K_2(p) \subseteq A$ (or both).
- For example, if atom A contains the formula $\neg\Diamond p$, it must also contain the formulae $\neg p$ and $\neg\bigcirc\Diamond p$.

Mutually Satisfiable Formulae

- A set of formulae $S \subseteq \Phi_\varphi$ is called mutually satisfiable if there exists a model σ and a position $j \geq 0$, such that every formula $p \in S$ holds at position j of σ.
- The intended meaning of an atom is that it represents a maximal mutually satisfiable set of formulae.

Claim 1 (atoms represent necessary conditions)
Let $S \subseteq \Phi_\varphi$ be a mutually satisfiable set of formulae.
Then there exists a φ-atom A such that $S \subseteq A$.

- It is important to realize that inclusion in an atom is only a necessary condition for mutual satisfiability (e.g., $\{\bigcirc p \vee \bigcirc\neg p, \bigcirc p, \bigcirc\neg p, p\}$ is an atom for the formula $\bigcirc p \vee \bigcirc\neg p$).

Basic Formulae

- A formula is called basic if it is either a proposition or has the form $\bigcirc p$, $\ominus p$, or $\circleddash p$.
- Basic formulae are important because their presence or absence in an atom uniquely determines all other closure formulae in the same atom.
- Let Φ_{φ}^{+} denote the set of formulae in Φ_{φ} that are not of the form $\neg\psi$.

Algorithm (atom construction)

1. Find all basic formulae $p_1, \cdots, p_b \in \Phi_{\varphi}^{+}$.
2. Construct all 2^b combinations.
3. Complete each combination into a full atom.

Example

- Consider the formula $\varphi_1 : \Box p \wedge \Diamond\neg p$ whose basic formulae are

$$p,\ \bigcirc\Box p,\ \bigcirc\Diamond\neg p.$$

- Following is the list of all atoms of φ_1:

$$\begin{array}{lllllll}
A_0: & \{\neg p, & \neg\bigcirc\Box p, & \neg\bigcirc\Diamond\neg p, & \neg\Box p, & \Diamond\neg p, & \neg\varphi_1\} \\
A_1: & \{p, & \neg\bigcirc\Box p, & \neg\bigcirc\Diamond\neg p, & \neg\Box p, & \neg\Diamond\neg p, & \neg\varphi_1\} \\
A_2: & \{\neg p, & \neg\bigcirc\Box p, & \bigcirc\Diamond\neg p, & \neg\Box p, & \Diamond\neg p, & \neg\varphi_1\} \\
A_3: & \{p, & \neg\bigcirc\Box p, & \bigcirc\Diamond\neg p, & \neg\Box p, & \Diamond\neg p, & \neg\varphi_1\} \\
A_4: & \{\neg p, & \bigcirc\Box p, & \neg\bigcirc\Diamond\neg p, & \neg\Box p, & \Diamond\neg p, & \neg\varphi_1\} \\
A_5: & \{p, & \bigcirc\Box p, & \neg\bigcirc\Diamond\neg p, & \Box p, & \neg\Diamond\neg p, & \neg\varphi_1\} \\
A_6: & \{\neg p, & \bigcirc\Box p, & \bigcirc\Diamond\neg p, & \neg\Box p, & \Diamond\neg p, & \neg\varphi_1\} \\
A_7: & \{p, & \bigcirc\Box p, & \bigcirc\Diamond\neg p, & \Box p, & \Diamond\neg p, & \varphi_1\}
\end{array}$$

The Tableau

- Given a formula φ, we construct a directed graph T_φ, called the tableau of φ, by the following algorithm.

Algorithm (tableau construction)

1. The nodes of T_φ are the atoms of φ.
2. Atom A is connected to atom B by a directed edge if all of the following are satisfied:
   - $R_{\bigcirc}$: For every $\bigcirc p \in \Phi_\varphi$, $\bigcirc p \in A$ iff $p \in B$.
   - $R_{\ominus}$: For every $\ominus p \in \Phi_\varphi$, $p \in A$ iff $\ominus p \in B$.
   - $R_{\circleddash}$: For every $\circleddash p \in \Phi_\varphi$, $p \in A$ iff $\circleddash p \in B$.

- An atom is called initial if it does not contain a formula of the form $\ominus p$ or $\neg \circleddash p$ ($\cong \ominus \neg p$).

Example

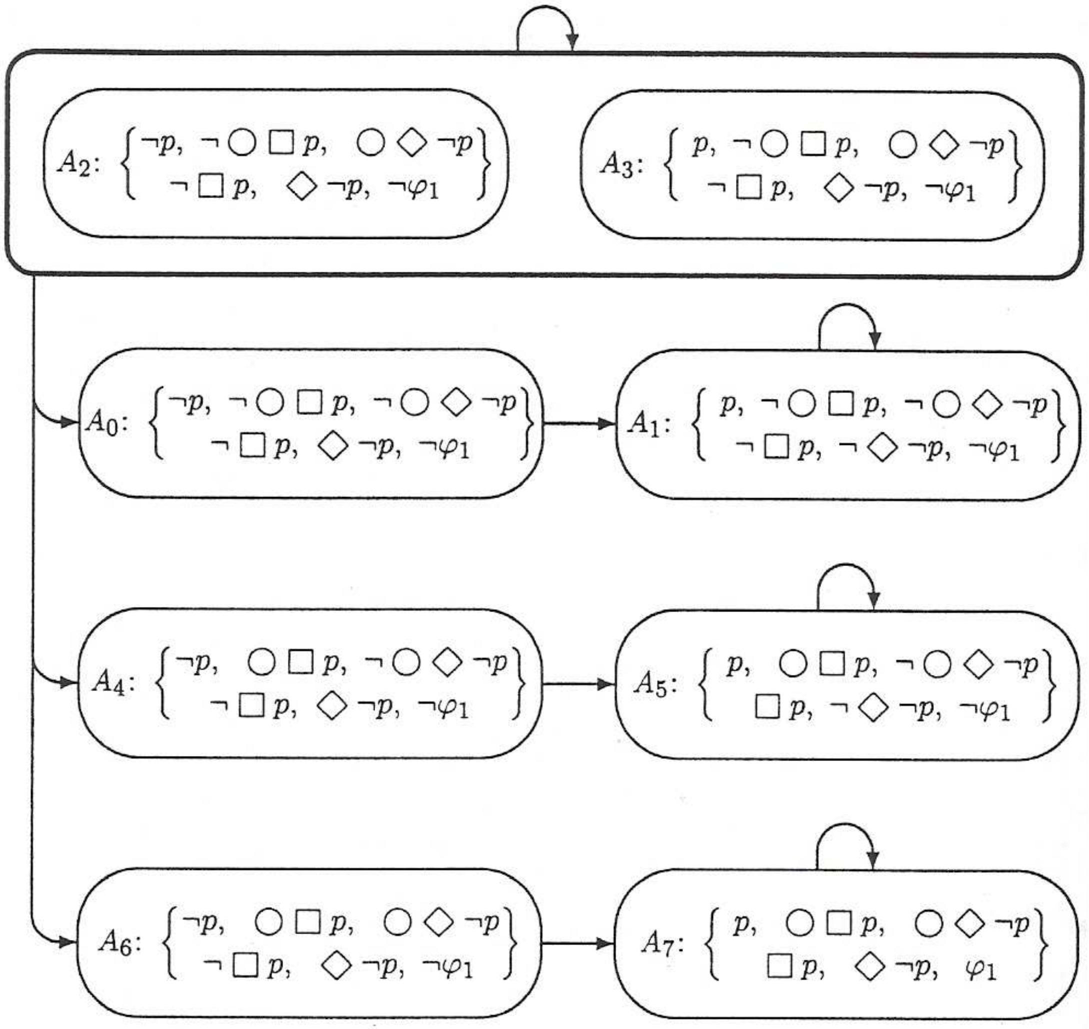


Fig. 5.3. Tableau T_{φ_1} for formula φ_1: $\Box p \wedge \Diamond \neg p$.

From the Tableau to a GBA

- Create an initial node and link it to every initial atom that contains φ.
- Label each directed edge with the atomic propositions that are contained in the ending atom.
- Add a set of atoms to the accepting set for each subformula of the following form:
  - $\Diamond q$: atoms with q or $\neg\Diamond q$.
  - $p\,\mathcal{U}\,q$: atoms with q or $\neg(p\,\mathcal{U}\,q)$.
  - $\neg\Box\neg q$ ($\cong \Diamond q$): atoms with q or $\Box\neg q$.
  - $\neg(\neg q\,\mathcal{W}\,p)$ ($\cong \neg p\,\mathcal{U}\,(q \wedge \neg p)$): atoms with q or $\neg q\,\mathcal{W}\,p$.
  - $\neg\Box q$ ($\cong \Diamond\neg q$): atoms with $\neg q$ or $\Box q$.
  - $\neg(q\,\mathcal{W}\,p)$ ($\cong \neg p\,\mathcal{U}\,(\neg q \wedge \neg p)$): atoms with $\neg q$ or $q\,\mathcal{W}\,p$.

Correctness: Models vs. Paths

- For a model σ, the infinite atom path $\pi_\sigma : A_0, A_1, \cdots$ in T_φ is said to be induced by σ if, for every position $j \geq 0$ and every closure formula $p \in \Phi_\varphi$,

$$(\sigma, j) \models p \text{ iff } p \in A_j.$$

Claim 2 (models induce paths)
Consider a formula φ and its tableau T_φ. For every model $\sigma : s_0, s_1, \cdots$, there exists an infinite atom path $\pi_\sigma : A_0, A_1, \cdots$ in T_φ induced by σ.

Furthermore, A_0 is an initial atom, and if $\sigma \models \varphi$ then $\varphi \in A_0$.

Correctness: Promising Formulae

- A formula $\psi \in \Phi_\varphi$ is said to promise the formula r if ψ has one of the following forms:

$$\Diamond r,\ p\,\mathcal{U}\,r,\ \neg\Box\neg r,\ \neg(\neg r\,\mathcal{W}\,p).$$

or if r is the negation $\neg q$ and ψ has one of the forms:

$$\neg\Box q,\ \neg(q\,\mathcal{W}\,p).$$

Claim 3 (promise fulfillment by models)
Let σ be a model and ψ, a formula promising r. Then, σ contains infinitely many positions $j \geq 0$ such that

$$(\sigma, j) \models \neg\psi \text{ or } (\sigma, j) \models r.$$

Correctness: Fulfilling Paths

- Atom A fulfills a formula ψ that promises r if $\neg\psi \in A$ or $r \in A$.
- A path $\pi : A_0, A_1, \cdots$ in the tableau T_φ is called fulfilling:
  - A_0 is an initial atom.
  - For every promising formula $\psi \in \Phi_\varphi$, π contains infinitely many atoms A_j that fulfill ψ.

Claim 4 (models induce fulfilling paths)
If $\pi_\sigma : A_0, A_1, \cdots$ is a path induced by a model σ, then π_σ is fulfilling.

Correctness: Fulfilling Paths (cont.)

Claim 5 (fulfilling paths induce models)
If $\pi : A_0, A_1, \cdots$ is a fulfilling path in T_φ, there exists a model σ inducing π, i.e., $\pi = \pi_\sigma$ and, for every $\psi \in \Phi_\varphi$ and every $j \geq 0$,

$$(\sigma, j) \models \psi \text{ iff } \psi \in A_j.$$

Proposition 6 (satisfiability and fulfilling paths)
Formula φ is satisfiable iff the tableau T_φ contains a fulfilling path $\pi = A_0, A_1, \cdots$ such that A_0 is an initial φ-atom.

Outline

- Büchi Automata
- Model Checking Using Automata
- Checking Emptiness
- Simple On-the-fly Translation
- Tableau Construction
- **Inductive Construction**

Inductive Construction

- We show how to construct a Büchi automaton inductively from a given temporal formula.
- The construction was originally proposed in [Kesten and Pnueli 2002] for proving completeness of a proof system for QPTL, the quantified version of PTL.
- Utilizing congruences on temporal formulae, the inductive step deals with the following cases:

$$\neg p, p \vee q, \bigcirc p, \Diamond p, \tilde{\ominus} p, \diamondminus p, \exists v : p.$$

  ($p\,\mathcal{U}\,q$ may be treated as $\exists t : t \wedge \Box(t \rightarrow q \vee (p \wedge \bigcirc t)) \wedge \neg\Box t$ and $p\,\mathcal{S}\,q$ as $\exists t : t \wedge \boxminus(t \rightarrow q \vee (p \wedge \ominus t))$.)

- The case of negation is rather involved and will be omitted.

Definitions

- We will use a slight variant of Büchi automaton.
- A Büchi automaton $\mathcal{A} = (Q, Q_0, \delta, F)$ consists of
  - Q: a finite set of automaton locations.
  - $Q_0 \subseteq Q$: a subset of initial automaton locations.
  - δ: for every $q_i, q_j \in Q$, $\delta(q_i, q_j)$ is a propositional assertion over $\mathcal{V}$ (a given set of Boolean variables).
  - F: a set of accepting locations.
- Let $\sigma = s_0, s_1, \cdots$ be a model, namely a sequence of truth assignments to $\mathcal{V}$.
- A sequence of automaton locations $\rho = q_0, q_1, \cdots$ is a run segment of $\mathcal{A}$ over σ, if $s_i \models \delta(q_i, q_{i+1})$, for every $i \geq 0$.
- A run segment $\rho = q_0, q_1, \cdots$ is a run of $\mathcal{A}$ if $q_0 \in Q_0$.

Definitions (cont.)

- A model σ is said to be accepted by the automaton $\mathcal{A}$, if $\mathcal{A}$ has an accepting run over σ.
- We denote by $\mathcal{L}(\mathcal{A})$, the set of all models accepted by $\mathcal{A}$.
- A model σ' is said to be a j-marked variant of σ if σ' is a t-variant of σ and σ' interprets t as T at position j and F elsewhere (t is a special variable in $\mathcal{V}$).
- Every model σ has a unique j-marked variant for each $j \geq 0$.
- Automaton $\mathcal{A}$ j-approves a model σ if it accepts the j-marked variant of σ.
- $\mathcal{A}$ is congruent to a formula φ not referring to t if, for every model σ and position $j \geq 0$, $(\sigma, j) \models \varphi$ iff $\mathcal{A}$ j-approves σ.

Case (Basis): p

Let p be a proposition and $\mathcal{A}_p = (Q, Q_0, \delta, F)$ be a Büchi automaton given by:

$$\begin{array}{lcl} Q & = & \{q_0, q_1\} \\ Q_0 & = & \{q_0\} \\ F & = & \{q_1\} \\ \delta(q_0, q_1) & = & p \wedge t \\ \delta(q_0, q_0) & = & \delta(q_1, q_1) \;=\; \neg t \\ \delta(q_1, q_0) & = & \mathsf{F} \end{array}$$

Claim: $\mathcal{A}_p$ is congruent to p.

Subsequently, we will use $\mathcal{A}_p = (Q^p, Q_0^p, \delta^p, F^p)$ to denote the Büchi automaton congruent to a formula p.

Case (Induction): $p \vee q$

The automaton $\mathcal{A}_{p\vee q} = (Q, Q_0, \delta, F)$ is given by:

$$\begin{aligned} Q &= Q^p \cup Q^q \\ Q_0 &= Q_0^p \cup Q_0^q \\ F &= F^p \cup F^q. \end{aligned}$$

For every $q_i, q_j \in Q$,

$$\delta(q_i, q_j) = \begin{cases} \delta^p(q_i, q_j) & \text{if } q_i, q_j \in Q^p \\ \delta^q(q_i, q_j) & \text{if } q_i, q_j \in Q^q \\ \text{F} & \text{otherwise}. \end{cases}$$

Case (Induction): $\bigcirc p$

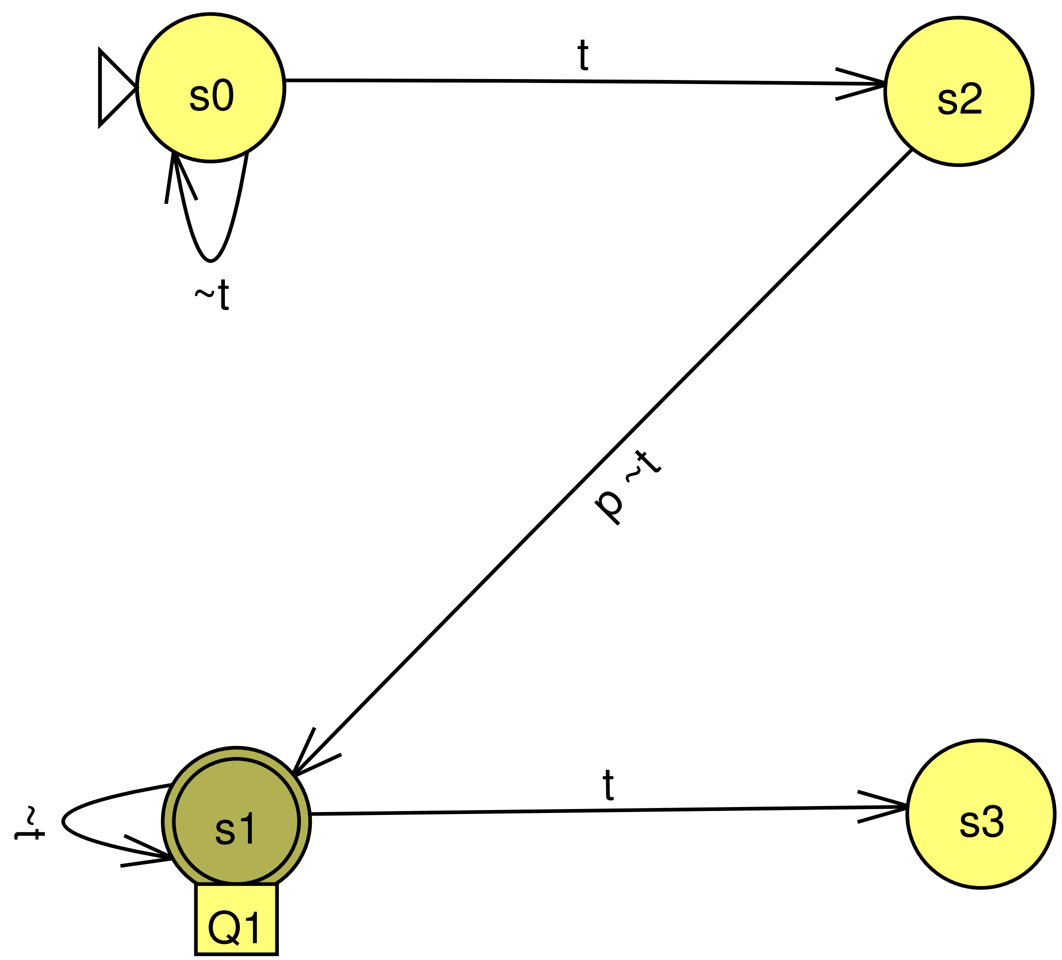
The automaton $\mathcal{A}_{\bigcirc p} = (Q, Q_0, \delta, F)$ is given by:

$$\begin{array}{rcl} Q & = & Q^p \cup (Q^p)' \\ Q_0 & = & Q_0^p \\ F & = & F^p. \end{array}$$

For every $q_i, q_j \in Q^p$, let $\delta^p(q_i, q_j) = \eta_{ij}(t)$. Then

$$\begin{array}{rcl} \delta(q_i, q_j) & = & \neg t \wedge \eta_{ij}[\mathsf{F}] \\ \delta(q_i, q_j') & = & t \wedge \eta_{ij}[\mathsf{F}] \\ \delta(q_i', q_j) & = & \neg t \wedge \eta_{ij}[\mathsf{T}] \\ \delta(q_i', q_j') & = & \mathsf{F}. \end{array}$$

Case (Induction): $\bigcirc p$ (cont.)



Case (Induction): $\ominus p$

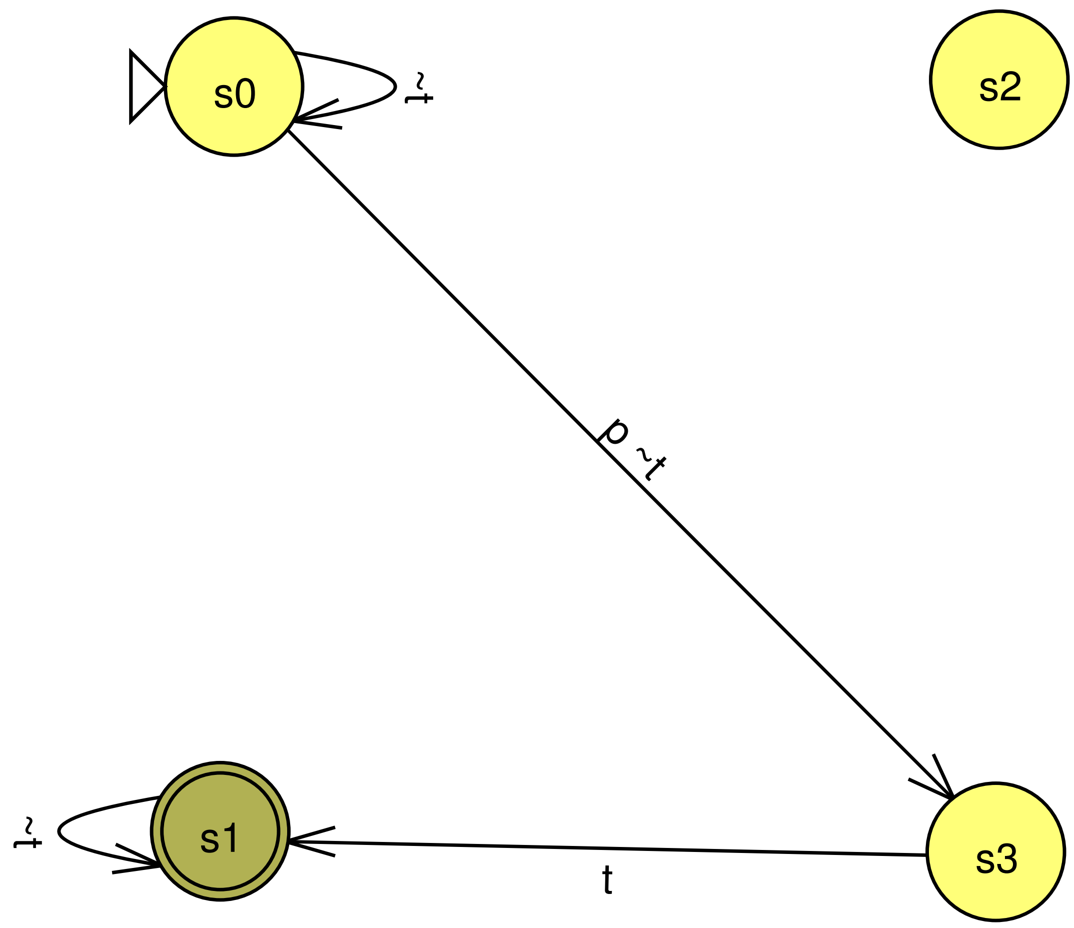
The automaton $\mathcal{A}_{\ominus p} = (Q, Q_0, \delta, F)$ is defined as follows:

$$\begin{array}{rcl} Q & = & Q^p \cup (Q^p)' \\ Q_0 & = & Q_0^p \\ F & = & F^p. \end{array}$$

For every $q_i, q_j \in Q^p$, let $\delta^p(q_i, q_j) = \eta_{ij}(t)$. Then

$$\begin{array}{rcl} \delta(q_i, q_j) & = & \neg t \wedge \eta_{ij}[\mathsf{F}] \\ \delta(q_i, q_j') & = & \neg t \wedge \eta_{ij}[\mathsf{T}] \\ \delta(q_i', q_j') & = & t \wedge \eta_{ij}[\mathsf{F}] \\ \delta(q_i', q_j) & = & \mathsf{F}. \end{array}$$

Case (Induction): $\ominus p$ (cont.)



Case (Induction): $\Diamond p$

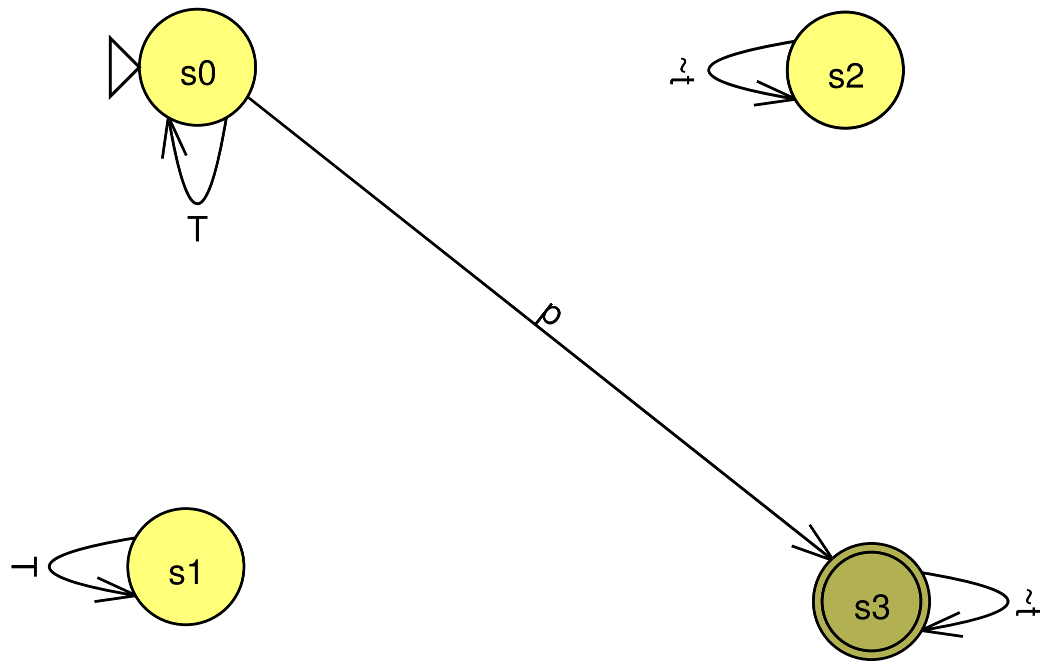
The automaton $\mathcal{A}_{\Diamond p} = (Q, Q_0, \delta, F)$ is defined as follows:

$$\begin{array}{rcl} Q & = & Q^p \cup (Q^p)' \\ Q_0 & = & Q_0^p \\ F & = & (F^p)'. \end{array}$$

For every $q_i, q_j \in Q^p$, let $\delta^p(q_i, q_j) = \eta_{ij}(t)$. Then

$$\begin{array}{rcl} \delta(q_i, q_j) & = & \eta_{ij}[\mathsf{F}] \\ \delta(q_i, q_j') & = & \eta_{ij}[\mathsf{T}] \\ \delta(q_i', q_j') & = & \neg t \wedge \eta_{ij}[\mathsf{F}] \\ \delta(q_i', q_j) & = & \mathsf{F}. \end{array}$$

Case (Induction): $\Diamond p$ (cont.)



Case (Induction): $\diamondminus p$

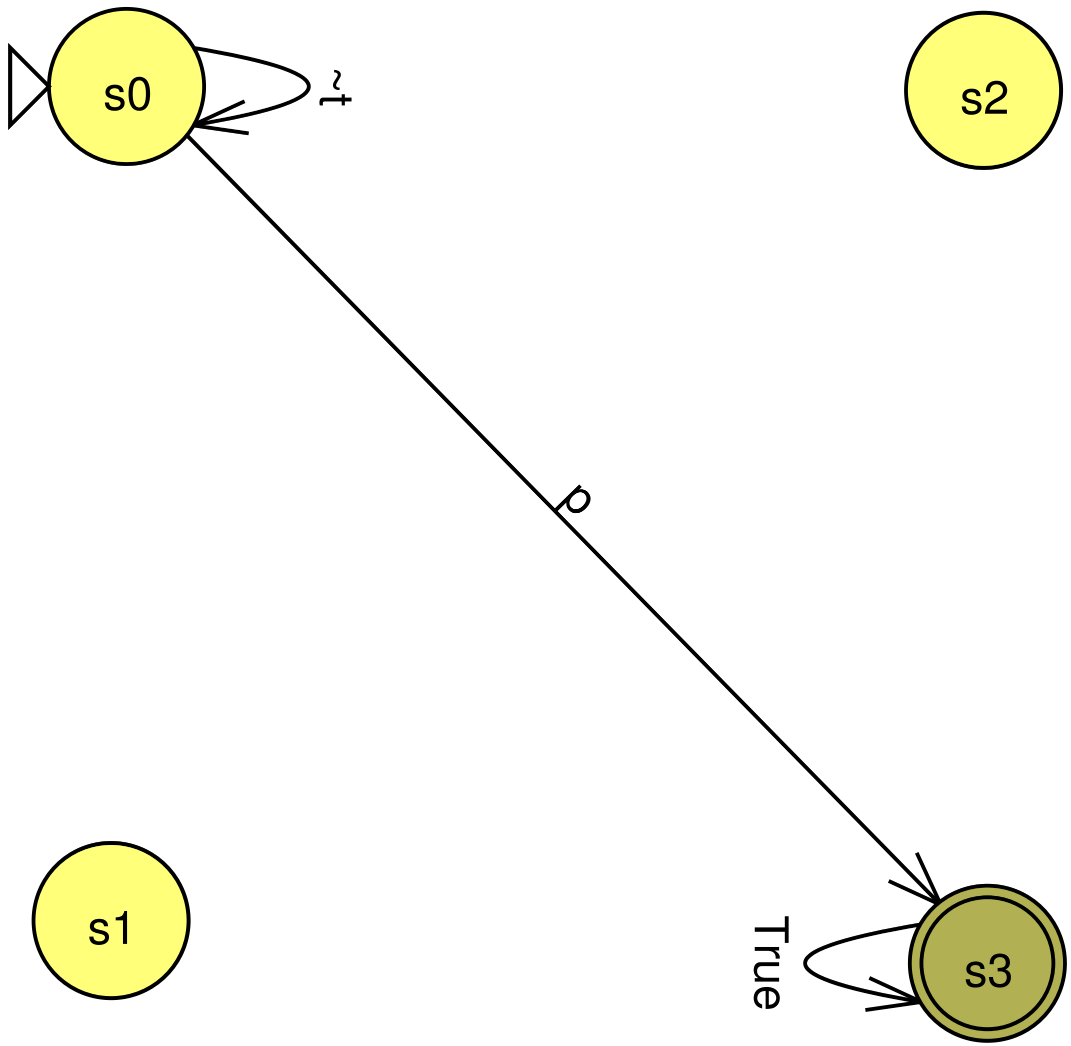
The automaton $\mathcal{A}_{\diamondminus p} = (Q, Q_0, \delta, F)$ is given by:

$$
\begin{aligned}
Q &= Q^p \cup (Q^p)' \\
Q_0 &= Q_0^p \\
F &= (F^p)'.
\end{aligned}
$$

For every $q_i, q_j \in Q^p$, let $\delta^p(q_i, q_j) = \eta_{ij}(t)$. Then

$$
\begin{aligned}
\delta(q_i, q_j) &= \neg t \wedge \eta_{ij}[\mathsf{F}] \\
\delta(q_i, q_j') &= \eta_{ij}[\mathsf{T}] \\
\delta(q_i', q_j') &= \eta_{ij}[\mathsf{F}] \\
\delta(q_i', q_j) &= \mathsf{F}.
\end{aligned}
$$

Case (Induction): $\diamondminus p$ (cont.)



Case (Induction): $\exists v : p$

For every $q_i, q_j \in Q^p$ and $v \in \mathcal{V} - \{t\}$, let $\delta^p(q_i, q_j) = \eta_{ij}(v)$. The automaton $\mathcal{A}_{\exists v:p} = (Q, Q_0, \delta, F)$ is defined as follows:

$$
\begin{array}{lcl}
Q & = & Q^p \\
Q_0 & = & Q_0^p \\
F & = & F^p \\
\delta(q_i, q_j) & = & \eta_{i,j}[\mathsf{F}] \vee \eta_{i,j}[\mathsf{T}]
\end{array}
$$