Verification – Lecture 11
Model Checking

Bernd Finkbeiner – Sven Schewe
Rayna Dimitrova – Lars Kuhtz – Anne Proetzsch

Wintersemester 2007/2008

REVIEW

From LTL to GNBA

GNBA $\mathcal{G}_\varphi$ over 2^{AP} for LTL-formula φ with $\mathcal{L}_\omega(\mathcal{G}_\varphi) = \mathit{Words}(\varphi)$:

- Assume φ only contains the operators $\wedge$, $\neg$, $\bigcirc$ and $\mathcal{U}$
  - $\vee$, $\rightarrow$, $\Diamond$, $\Box$, $\mathcal{W}$, and so on, are expressed in terms of these basic operators
- States are *elementary sets* of sub-formulas in φ
  - for $\sigma = A_0 A_1 A_2 \ldots \in \mathit{Words}(\varphi)$, expand $A_i \subseteq AP$ with sub-formulas of φ
  - ... to obtain the infinite word $\bar{\sigma} = B_0 B_1 B_2 \ldots$ such that

$$\psi \in B_i \quad \text{if and only if} \quad \sigma^i = A_i A_{i+1} A_{i+2} \ldots \models \psi$$

  - $\bar{\sigma}$ is intended to be a run in GNBA $\mathcal{G}_\varphi$ for σ
- Transitions are derived from semantics $\bigcirc$ and expansion law for $\mathcal{U}$
- Accept sets guarantee that: $\bar{\sigma}$ is an accepting run for σ iff $\sigma \models \varphi$

Elementary sets of formulae

$B \subseteq$ *closure*(φ) is *elementary* if:

1. B is *logically consistent* if for all $\varphi_1 \wedge \varphi_2, \psi \in$ *closure*(φ):
   - $\varphi_1 \wedge \varphi_2 \in B \;\Leftrightarrow\; \varphi_1 \in B$ and $\varphi_2 \in B$
   - $\psi \in B \;\Rightarrow\; \neg\psi \notin B$
   - true $\in$ *closure*$(\varphi) \;\Rightarrow\;$ true $\in B$
2. B is *locally consistent* if for all $\varphi_1 \mathcal{U} \varphi_2 \in$ *closure*(φ):
   - $\varphi_2 \in B \;\Rightarrow\; \varphi_1 \mathcal{U} \varphi_2 \in B$
   - $\varphi_1 \mathcal{U} \varphi_2 \in B$ and $\varphi_2 \notin B \;\Rightarrow\; \varphi_1 \in B$
3. B is *maximal*, i.e., for all $\psi \in$ *closure*(φ):
   - $\psi \notin B \;\Rightarrow\; \neg\psi \in B$

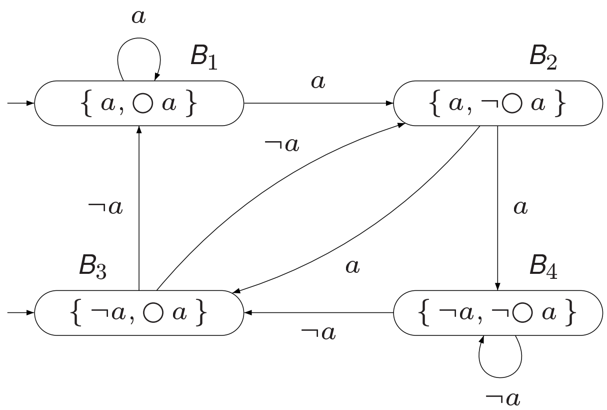
The GNBA of LTL-formula φ

For LTL-formula φ, let $\mathcal{G}_\varphi = (Q, 2^{AP}, \delta, Q_0, \mathcal{F})$ where

- Q = all elementary sets $B \subseteq$ *closure*(φ) , $Q_0 = \{ B \in Q \mid \varphi \in B \}$
- $\mathcal{F} \;=\; \big\{ \{ B \in Q \mid \varphi_1 \mathcal{U} \varphi_2 \notin B$ or $\varphi_2 \in B \} \mid \varphi_1 \mathcal{U} \varphi_2 \in$ *closure*$(\varphi) \big\}$
- The transition relation $\delta : Q \times 2^{AP} \rightarrow 2^Q$ is given by:
  - If $A \neq B \cap AP$ then $\delta(B, A) \;=\; \varnothing$
  - $\delta(B, B \cap AP)$ is the set of all elementary sets of formulas B' satisfying:
    (i) For every $\bigcirc \psi \in$ *closure*(φ): $\bigcirc \psi \in B \;\Leftrightarrow\; \psi \in B'$, and
    (ii) For every $\varphi_1 \mathcal{U} \varphi_2 \in$ *closure*(φ):

$$\varphi_1 \mathcal{U} \varphi_2 \in B \;\Leftrightarrow\; \Big(\varphi_2 \in B \vee (\varphi_1 \in B \;\wedge\; \varphi_1 \mathcal{U} \varphi_2 \in B')\Big)$$

GNBA for LTL-formula $\bigcirc a$



$Q_0 = \{ B_1, B_3 \}$ since $\bigcirc a \in B_1$ and $\bigcirc a \in B_3$

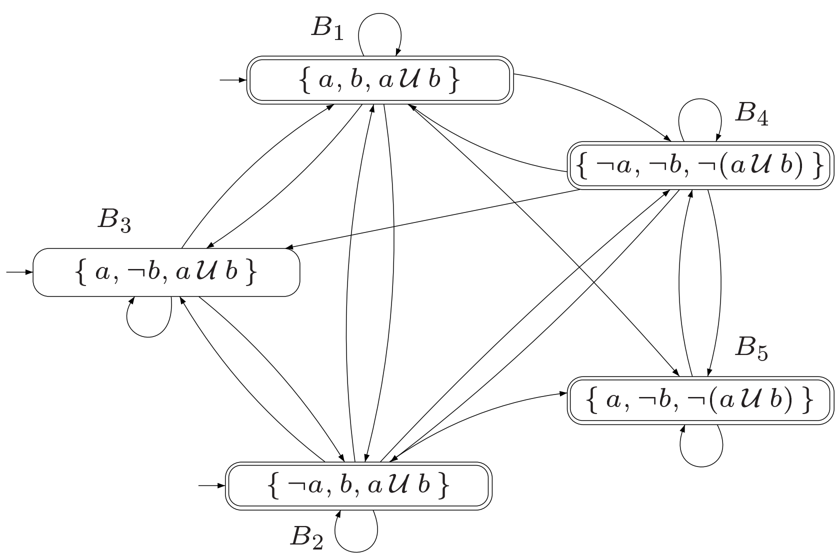
$\delta(B_2, \{ a \}) = \{ B_3, B_4 \}$ as $B_2 \cap \{ a \} = \{ a \}, \neg\bigcirc a = \bigcirc \neg a \in B_2$, and $\neg a \in B_3, B_4$

$\delta(B_1, \{ a \}) = \{ B_1, B_2 \}$ as $B_1 \cap \{ a \} = \{ a \}, \bigcirc a \in B_1$ and $a \in B_1, B_2$

$\delta(B_4, \{ a \}) = \varnothing$ since $B_4 \cap \{ a \} = \varnothing \neq \{ a \}$

The set $\mathcal{F}$ is empty, since $\varphi = \bigcirc a$ does not contain an until-operator

GNBA for LTL-formula $a \,\mathcal{U}\, b$



Correctness theorem

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

$$\textit{Words}(\varphi) = \{\sigma \in \Sigma^\omega \mid \sigma \models \varphi\}$$

NBA are more expressive than LTL

Corollary: every LTL-formula expresses an ω-regular property

But: there exist ω-regular properties that cannot be expressed in LTL

Example: there is no LTL formula φ with *Words*$(\varphi) = P$ for the LT-property:

$$P = \left\{ A_0 A_1 A_2 \ldots \in \left(2^{\{a\}}\right)^\omega \mid a \in A_{2i} \text{ for } i \geqslant 0 \right\}$$

But there exists an NBA $\mathcal{A}$ with $\mathcal{L}_\omega(\mathcal{A}) = P$

$\Rightarrow$ *there are ω-regular properties that cannot be expressed in LTL!*

Complexity for LTL to NBA

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

Time and space complexity in $2^{\mathcal{O}(|\varphi| \cdot \log |\varphi|)}$

- States GNBA $\mathcal{G}_\varphi$ are elementary sets of formulae in *closure*(φ)
  - sets B can be represented by bit vectors with single bit per subformula ψ of φ
- The number of states in $\mathcal{G}_\varphi$ is bounded by $2^{|\text{subf}(\varphi)|}$
  - where subf(φ) denotes the set of all subformulae of φ
- The number of accepting sets of $\mathcal{G}_\varphi$ is bounded above by $\mathcal{O}(|\varphi|)$
- The number of states in NBA $\mathcal{A}_\varphi$ is thus bounded by $2^{\mathcal{O}(|\varphi)|} \cdot \mathcal{O}(|\varphi|)$
- $2^{\mathcal{O}(|\varphi|)} \cdot \mathcal{O}(|\varphi|) = 2^{\mathcal{O}(|\varphi|)}$ ■

Lower bound

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

Proof (1)

Let AP be non-empty, that is, $|2^{AP}| \geqslant 2$ and:

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

It follows $\mathcal{L}_n = Words(\varphi_n)$ where $\varphi_n = \bigwedge_{a \in AP} \bigwedge_{0 \leqslant i < n} (\bigcirc^i a \longleftrightarrow \bigcirc^{n+i} a)$

φ_n is an LTL formula of polynomial length: $|\varphi_n| \in \mathcal{O}\Big(|AP| \cdot n\Big)$

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

Proof (2)

Claim: any NBA $\mathcal{A}$ for $\bigwedge_{a \in AP} \bigwedge_{0 \leqslant i < n} (\bigcirc^i a \longleftrightarrow \bigcirc^{n+i} a)$ has at least 2^n states

Words of the form $A_1 \ldots A_n\, A_1 \ldots A_n\, \varnothing\, \varnothing\, \varnothing \ldots$ are accepted by $\mathcal{A}$

$\mathcal{A}$ thus has for every word $A_1 \ldots A_n$ of length n, a state $q(A_1 \ldots A_n)$, say, which can be reached from an initial state by consuming $A_1 \ldots A_n$

From $q(A_1 \ldots A_n)$, it is possible to visit an accept state infinitely often by accepting the suffix $A_1 \ldots A_n\, \varnothing\, \varnothing\, \varnothing \ldots$

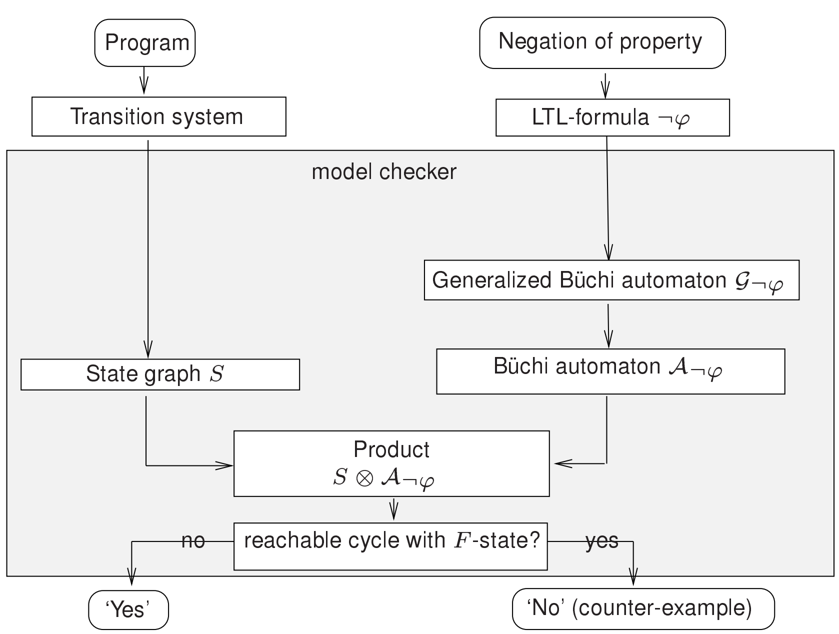
If $A_1 \ldots A_n \neq A'_1 \ldots A'_n$ then

$$A_1 \ldots A_n\, A'_1 \ldots A'_n\, \varnothing\, \varnothing\, \varnothing \ldots \notin \mathcal{L}_n = \mathcal{L}_\omega(\mathcal{A})$$

Therefore, the states $q(A_1 \ldots A_n)$ are all pairwise different

Given $|2^{AP}|$ possible sequences $A_1 \ldots A_n$, NBA $\mathcal{A}$ has $\geqslant \left(\left|2^{AP}\right|\right)^n \geqslant 2^n$ states

LTL model checking



Fair Transition Systems

$$\Phi = (V, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C})$$

- $\mathcal{J} \subseteq \mathcal{T}$: set of just (weakly fair) transitions.
- $\mathcal{C} \subseteq \mathcal{T}$: set of compassionate (strongly fair) transitions.

- Justice: for each just transition it is not the case that the transition is continually enabled but only taken at finitely many positions.
- Compassion: for each compassionate transition it is not the case that the transition is enabled at infinitely many positions but only taken at finitely many positions.

Fairness

- Justice can be specified in LTL as follows:

$$\textit{justice} = \bigwedge_{\tau \in \mathcal{J}} (\Box\, \textit{enabled}(\tau)) \Rightarrow (\Box \Diamond\, \textit{taken}(\tau))$$

- Compassion can be specified in LTL as follows:

$$\textit{compassion} = \bigwedge_{\tau \in \mathcal{C}} (\Box \Diamond\, \textit{enabled}(\tau)) \Rightarrow (\Box \Diamond\, \textit{taken}(\tau))$$

Fairness

Verification of fair transition systems can be reduced to the verification of transition systems without fairness:

- Let *fair* $=$ *justice* $\wedge$ *compassion*.
- Then,

$$(V, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C}) \models \varphi \quad \text{iff} \quad (V, \Theta, \mathcal{T}, \emptyset, \emptyset) \models \textit{fair} \rightarrow \varphi$$

Cycle detection

How to check for a reachable cycles containing an F-state?

- Alternative 1:
  - compute the strongly connected components (SCCs) in $G(S)$
  - check whether one such SCC is reachable from an initial state
  - . . . that contains an F-state
  - "eventually forever $\neg F$" is refuted if and only if such SCC is found
- Alternative 2:
  - *use a nested depth-first search*

  $\Rightarrow$ more adequate for an on-the-fly verification algorithm

  $\Rightarrow$ easier for generating counterexamples

let's have a closer look into this by first dealing with two-phase DFS

A two-phase depth first-search

1. Determine all F-states that are reachable from some initial state

   this is performed by a standard depth-first search

2. For each reachable F-state, check whether it belongs to a cycle

   – start a depth-first search in s
   – check for all states reachable from s whether there is a "backward" edge to s

- Time complexity: $\Theta(N \cdot (N+M))$

  – where N is the number of states and M the number of edges
  – fragments reachable via K F-states are searched K times

Two-phase depth first-search

Input: finite-state transition system S and accept set F
Output: "yes" if S contains a reachable cycle with an F-state, otherwise "no".

---

set of states $R := \varnothing$; $R_F := \varnothing$; (* set of reachable states resp. F-states *)
stack of states $U := \varepsilon$; (* DFS-stack for first DFS, initial empty *)
set of states $T := \varnothing$; (* set of visited states for the cycle check *)
stack of states $V := \varepsilon$; (* DFS-stack for the cycle check *)

for all $s \models \Theta$ and $s \notin R$ **do** visit(s); **od** (* phase one *)
for all $s \in R_F$ **do**
  $T := \varnothing$; $V := \varepsilon$; (* phase two *)
  if cycle_check(s) **then** return "no" (* s belongs to a cycle *)
od
return "yes" (* none of the F-states belongs to a cycle *)

Find F-states

```
procedure visit (state s)
  push(s, U);                                         (* push s on the stack *)
  R := R ∪ { s };                                     (* mark s as reachable *)
  repeat
    s' := top(U);
    if Successors(s') ⊆ R then
      pop(U);
      if s' ∈ F then R_F := R_F ∪ { s' }; fi
    else
      let s'' ∈ Successors(s') \ R
      push(s'', U);
      R := R ∪ { s'' };                    (* state s'' is a new reachable state *)
    fi
  until (U = ε)
endproc
```

this is a standard DFS checking for F-states

Cycle detection

```
procedure boolean cycle_check(state s)
  boolean cycle_found := false;                       (* no cycle found yet *)
  push(s, V); T := T ∪ { s };                         (* push s on the stack *)
  repeat
    s' := top(V);                                     (* take top element of V *)
    if s ∈ Successors(s') then
      cycle_found := true;            (* if s ∈ Successors(s'), a cycle is found *)
      push(s, V);                                     (* push s on the stack *)
    else
      if Successors(s') \ T ≠ ∅ then
        let s'' ∈ Successors(s') \ T;
        push(s'', V); T := T ∪ { s'' };    (* push an unvisited successor of s' *)
        else pop(V);                       (* unsuccessful cycle search for s' *)
      fi
    fi
  until ((V = ε) ∨ cycle_found)
  return cycle_found
endproc
```

Nested depth-first search

- Idea: perform the two depth-first searches in an *interleaved* way
  - the outer DFS serves to encounter all reachable F-states
  - the inner DFS seeks for backward edges
- *Nested DFS*
  - on full expansion of F-state s in the outer DFS, start inner DFS
  - in inner DFS, visit all states reachable from s *not visited* in the inner DFS yet
  - no backward edge found? continue the outer DFS (look for next F state)
- *Counterexample generation*: DFS stack concatenation
  - stack U for the outer DFS = path fragment from $s_0 \in I$ to s (in reversed order)
  - stack V for the inner DFS = a cycle from state s to s (in reversed order)

The outer DFS (1)

Input: transition system S without terminal states, and proposition Φ
Output: "yes" if S contains a reachable cycle with an F-state, otherwise "no" plus counterexample

---

set of states $R := \varnothing$; (* set of visited states in the outer DFS *)
stack of states $U := \varepsilon$; (* stack for the outer DFS *)
set of states $T := \varnothing$; (* set of visited states in the inner DFS *)
stack of states $V := \varepsilon$; (* stack for the inner DFS *)
boolean *cycle_found* := false;

while $(I \setminus R \neq \varnothing \ \wedge \ \neg$*cycle_found*) **do**
  let $s \in I \setminus R$; (* explore the reachable *)
  reachable_cycle(s); (* fragment with outer DFS *)
od
if $\neg$*cycle_found* **then**
  return ("yes")
else
  return ("no", *reverse*($V.U$)) (* stack contents yield a counterexample *)
fi

The outer DFS (2)

```
procedure reachable_cycle (state s)
  push(s, U);                                   (* push s on the stack *)
  R := R ∪ { s };
  repeat
    s' := top(U);
    if Successors(s') \ R ≠ ∅ then
      let s'' ∈ Successors(s') \ R;
      push(s'', U);                             (* push the unvisited successor of s' *)
      R := R ∪ { s'' };                         (* and mark it reachable *)
    else
      pop(U);                                   (* outer DFS finished for s' *)
      if s' ∈ F then
        cycle_found := cycle_check(s');         (* proceed with the inner *)
                                                (* DFS in state s' *)
      fi
    fi
  until ((U = ε) ∨ cycle_found)                 (* stop when stack for the outer *)
                                                (* DFS is empty or cycle found *)
endproc
```


Time complexity

The worst-case time complexity of nested DFS is in

$$\mathcal{O}(N+M)$$

where N is # reachable states in S, and M is # edges in state graph