Verification

Lecture 14

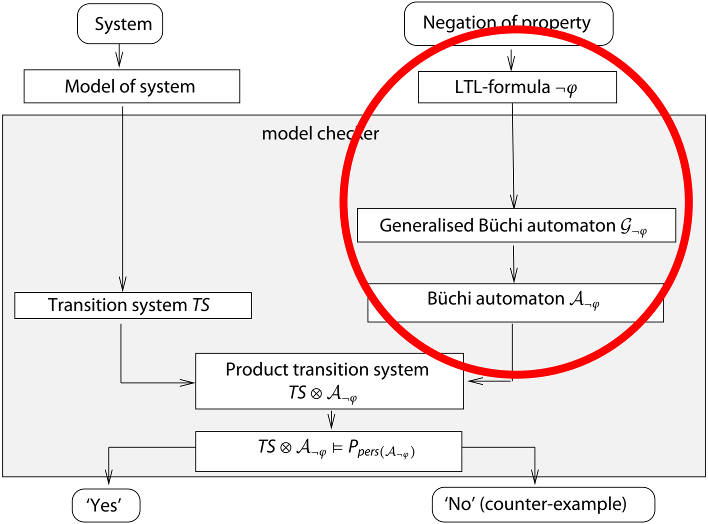
Bernd Finkbeiner

UNIVERSITÄT
DES
SAARLANDES

Plan for today

- LTL model checking (continued)
  - LTL vs. NBA
  - Persistency checking via nested DFS

Overview of LTL model checking



REVIEW: Main result

[Vardi, Wolper & Sistla 1986]

For any LTL-formula φ (over *AP*) there exists a GNBA $\mathcal{G}_\varphi$ over 2^{AP} such that:

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

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

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

$\Rightarrow$ every LTL-formula expresses an ω-regular property!

NBA are more expressive than LTL

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

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

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!

Proof

- Proof by contradiction:
  Assume there is an LTL formula φ with $Words(\varphi) = P$.
- Let $w_1 = \{a\}^{n+1}\varnothing\{a\}^{\omega}$ and
  $w_2 = \{a\}^{n+2}\varnothing\{a\}^{\omega}$
  where n is the number of $\bigcirc$-operators in φ.
  We show that $w_1 \in Words(\varphi)$ iff $w_2 \in Words(\varphi)$.
  This contradicts $Words(\varphi) = P$.
  Structural induction on φ:
- $\varphi \in AP$: φ only depends on first position
- $\varphi = \bigcirc\psi$: by IH, $\{a\}^{n}\varnothing\{a\}^{\omega} \in Words(\psi)$ iff
  $\{a\}^{n+1}\varnothing\{a\}^{\omega} \in Words(\psi)$.
  Hence, $w_1 \in Words(\varphi)$ iff $w_2 \in Words(\varphi)$.

Proof (cont'd)

- $\varphi = \psi_1 \, \mathsf{U} \, \psi_2$:
  1. $w_1 \in \mathit{Words}(\varphi) \Rightarrow w_2 \in \mathit{Words}(\varphi)$:
     - Case 1: $w_1 \models \psi_2$. Then, by IH, $w_2 \models \psi_2$.
     - Case 2: $w_1 \not\models \psi_2$. Let k be the smallest index such that $w_1[k \ldots] \models \psi_2$ and $\forall 0 \leq i < k. w_1[i \ldots] \models \psi_1$.
       $\Rightarrow w_2[k+1, \ldots] \models \psi_2$ and $\forall 1 \leq i < k. w_2[i \ldots] \models \psi_1$.
       Additionally, by IH, $w_1 \models \psi_1 \Rightarrow w_2 \models \psi_1$.
  2. $w_2 \in \mathit{Words}(\varphi) \Rightarrow w_1 \in \mathit{Words}(\varphi)$
     - Case 1: $w_2 \models \psi_2$. Then, by IH, $w_1 \models \psi_2$.
     - Case 2: $w_2 \not\models \psi_2$. Let k be the smallest index such that $w_2[k \ldots] \models \psi_2$ and $\forall 0 \leq i < k. w_2[i \ldots] \models \psi_1$.
       $\Rightarrow w_1[k-1, \ldots] \models \psi_2$ and $\forall 0 \leq i < k-1. w_1[i \ldots] \models \psi_1$.

Complexity of LTL-to-NBA translation

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 $2^{\mathcal{O}(|\varphi|)}$

Justification complexity: next slide

Time and space complexity

- 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 $2^{|\varphi|}$.
- The number of accepting sets of $\mathcal{G}_\varphi$ is bounded by $|\varphi|$.
- The number of states in NBA $\mathcal{A}_\varphi$ is thus bounded by $2^{|\varphi|} \cdot |\varphi| = 2^{(|\varphi| + \log |\varphi|)} = 2^{\mathcal{O}(|\varphi|)}$.

qed

Lower bound

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

Proof

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

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

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

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 (cont'd)

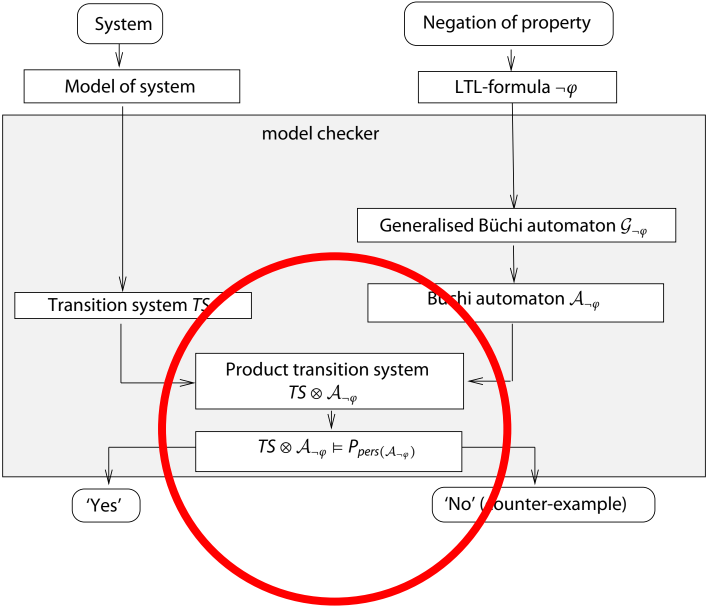
Claim: any NBA $\mathcal{A}$ for $\bigwedge_{a \in AP} \bigwedge_{0 \leq i < n} (\bigcirc^i a \longleftrightarrow \bigcirc^{n+i} a)$ has at least 2^n states

- 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 $q(A_1 \ldots A_n)$, 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 $\geq \left(|2^{AP}|\right)^n \geq 2^n$ states

Overview of LTL model checking



Synchronous product

For transition system $TS = (S, Act, \rightarrow, I, AP, L)$ without terminal states and $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ a non-blocking NBA with $\Sigma = 2^{AP}$, let:

$$TS \otimes \mathcal{A} \;=\; (S', Act, \rightarrow', I', AP', L') \qquad \text{where}$$

- $S' = S \times Q$, $AP' = Q$ and $L'(\langle s, q \rangle) \;=\; \{\, q \,\}$
- $\rightarrow'$ is the smallest relation defined by: $\dfrac{s \xrightarrow{\alpha} t \;\wedge\; q \xrightarrow{L(t)} p}{\langle s, q \rangle \xrightarrow{\alpha}{}' \langle t, p \rangle}$
- $I' = \{\, \langle s_0, q \rangle \mid s_0 \in I \;\wedge\; \exists q_0 \in Q_0.\, q_0 \xrightarrow{L(s_0)} q \,\}$

REVIEW: Reduction to persistence checking

$TS \models \varphi$ if and only if $Traces(TS) \subseteq Words(\varphi)$

if and only if $Traces(TS) \cap \left((2^{AP})^\omega \setminus Words(\varphi)\right) = \varnothing$

if and only if $Traces(TS) \cap \underbrace{Words(\neg\varphi)}_{\mathcal{L}_\omega(\mathcal{A}_{\neg\varphi})} = \varnothing$

if and only if $TS \otimes \mathcal{A}_{\neg\varphi} \models \Diamond \Box \neg F$

LTL model checking is thus reduced to persistence checking!

On-the-fly LTL model checking

- Idea: find a counter-example during the generation of $Reach(TS)$ and $\mathcal{A}_{\neg\varphi}$
  - exploit the fact that $Reach(TS)$ and $\mathcal{A}_{\neg\varphi}$ can be generated in parallel

$\Rightarrow$ Generate $Reach(TS \otimes \mathcal{A}_{\neg\varphi})$ "on demand"

  - consider a new vertex only if no accepting cycle has been found yet
  - only consider the successors of a state in $\mathcal{A}_{\neg\varphi}$ that match current state in TS

$\Rightarrow$ Possible to find an accepting cycle without generating $\mathcal{A}_{\neg\varphi}$ entirely

- This on-the-fly scheme is adopted for example in the model checker SPIN

Cycle detection

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

- Alternative 1: ($\rightarrow$ unconditional fairness in Lecture 4)
  - compute the strongly connected components (SCCs)
  - check whether one such SCC is reachable from an initial state
  - . . . that contains an *F*-state
- Alternative 2:
  - use a nested depth-first search

  $\Rightarrow$ adequate for on-the-fly verification

  $\Rightarrow$ easier for generating counterexamples

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 an "backward" edge to s

- Quadratic complexity

Two-phase depth first-search

Require: finite transition system *TS* without terminal states, and proposition F

Ensure: "yes" if $TS \models \Diamond \Box \neg F$", 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 $s \in I \setminus R$ **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}
end for
return "yes" {none of the F-states belongs to a cycle}

Find F-states

process visit (state s)
$push(s, U)$; {push s on the stack}
$R := R \cup \{ s \}$; {mark s as reachable}
repeat
    $s' := top(U)$;
    if $Post(s') \subseteq R$ **then**
        $pop(U)$;
        if $s' \models F$ **then** $R_F := R_F \cup \{ s' \}$; **fi**
    else
        let $s'' \in Post(s') \setminus R$
        $push(s'', U)$;
        $R := R \cup \{ s'' \}$; {state s'' is a new reachable state}
    end if
until $(U = \varepsilon)$ **endproc**

this is standard DFS checking for F-states

Cycle detection

process boolean cycle_check(state s)
 boolean *cycle_found* := false; {no cycle found yet}
 $push(s, V)$; $T := T \cup \{ s \}$; {push s on the stack}
 repeat
    $s' := top(V)$; {take top element of V}
    if $s \in Post(s')$ **then**
        cycle_found := true; {if $s \in Post(s')$, a cycle is found}
        $push(s, V)$; {push s on the stack}
    else
        if $Post(s') \setminus T \neq \varnothing$ **then**
            let $s'' \in Post(s') \setminus T$;
            $push(s'', V)$; $T := T \cup \{ s'' \}$; {push an unvisited successor of s'}
            else $pop(V)$; {unsuccessful cycle search for s'}
        end if
    end if
 until $((V = \varepsilon) \vee cycle\_found)$
 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 leading to the F-state
- 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 to s? 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)

Require: transition system TS without terminal states, and proposition F
Ensure: "yes" if $TS \models \Diamond \Box \neg F$, 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}
end while
if $\neg cycle\_found$ **then**
    return ("yes") {$TS \models \Diamond \Box \neg F$}
else
    return ("no", $reverse(V.U)$) {stack contents yield a counterexample}
end if

The outer DFS (2)

process reachable_cycle (state s)
$push(s, U)$; {push s on the stack}
$R := R \cup \{ s \}$;
repeat
    $s' := top(U)$;
    if $Post(s') \setminus R \neq \varnothing$ **then**
        let $s'' \in Post(s') \setminus R$;
        $push(s'', U)$; {push the unvisited successor of s'}
        $R := R \cup \{ s'' \}$; {and mark it reachable}
    else
        $pop(U)$; {outer DFS finished for s'}
        if $s' \models F$ **then**
            $cycle\_found :=$ cycle_check(s'); {proceed with the inner DFS in state s'}
        end if
    end if
until $((U = \varepsilon) \vee cycle\_found)$ {stop when stack for the outer DFS is empty or cycle found}
endproc

Correctness of nested DFS

Let:

- *TS* be a finite transition system over *AP* without terminal states and
- $\Diamond \Box \neg F$ a persistence property

$\boxed{\text{The nested DFS algorithm yields "no" if and only if } TS \not\models \Diamond \Box \neg F}$