Verification

Lecture 15

Bernd Finkbeiner

UNIVERSITÄT
DES
SAARLANDES

Plan for today

- Complexity of LTL model checking
- Bounded model checking

The LTL model-checking problem is co-NP-hard

The Hamiltonian path problem is polynomially reducible to the complement of the LTL model-checking problem

In fact, the LTL model-checking problem is PSPACE-complete

[Sistla & Clarke 1985]

Reduction from Hamiltonian Path Problem

- Hamiltonian Path for a directed graph (V, E) passes every vertex exactly once.
- The Hamiltonion Path Problem "Does a given graph have a Hamiltonian Path?" is NP-complete.
- The Hamiltonian Path Problem is polynomially reducible to the complement of the LTL model checking problem.
- Transition system: $S = V \cup \{b\}$; $\rightarrow = E \cup (V \cup \{b\}) \times \{b\}$; $L(v) = \{v\}$ for $v \in V$, $L(b) = \varnothing$
- LTL property "no path is Hamiltonian":

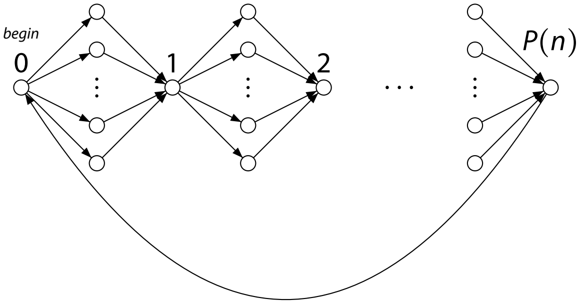
$$\neg \bigwedge_{v \in V} (\Diamond v \quad \wedge \quad \Box(v \rightarrow \bigcirc \Box \neg v))$$

PSPACE-hardness

- Let M be a polynomial space-bounded Turing machine that accepts words of a language K (i.e., K is a PSPACE-language)
- We construct for each word w a transition system TS and an LTL formula φ such that $TS \vDash \varphi$ iff $w \in K$.

Single-tape Turing machine $(Q, q_0, F, \Sigma, \delta)$
$\delta : Q \times \Sigma \rightarrow Q \times \Sigma \times \{L, R, N\}$
L: left, R: right, N: no move

Space-bounded: there is a polynomial $P(n)$ such that the computation on input word of length n visits at most $P(n)$ tape cells.



$$S = \{0, 1, \ldots, P(n)\} \cup \{(q, A, i) \mid q \in Q \cup \{*\}, A \in \Sigma, 0 < i \leq P(n)\}$$

Idea: $q \in Q$ identifies current state of Turing machine and current position of cursor; $*$ everywhere else.

- Configuration (Tape content $A_1, \ldots, A_{P(n)}$, current state q, cursor position i)
  is encoded as path fragment
  $0(*, A_1, 1)1(*, A_2, 2)2 \ldots i-1(q, A_i, i)i(*, A_{i+1}, i+1) \ldots P(n)$
- Computation is encoded as a sequence of such fragments.
- Legal configurations:
  $\varphi_{conf} = \Box\,(begin \rightarrow \varphi^1_{conf} \wedge \varphi^2_{conf})$
  $\varphi^1_{conf} = \bigvee_{1 \leq i \leq P(n)} \bigcirc^{2i-1} \Phi_Q$ where $\Phi_Q = \bigvee_{(q,A,i) \in S, q \in Q} (q, A, i)$
  $\varphi^2_{conf} = \bigwedge_{1 \leq i \leq P(n)} (\bigcirc^{2i-1} \Phi_Q \rightarrow \bigwedge_{1 \leq j \leq P(n), j \neq i} \bigcirc^{2j-1} \neg \Phi_Q)$

Transition function

for $\delta(q, A) = (p, B, L)$:

$$\varphi_{q,A} = \Box\,(begin \to \bigwedge_{1 \le i \le P(n)} (\bigcirc^{2i-1}(q, A, i) \to \psi(q, A, i, p, B, L)))$$

where

$$\psi(q, A, i, p, B, L) = \underbrace{\bigwedge_{1 \le j \le P(n), i \ne j, C \in \Sigma} (\bigcirc^{2j-1} C \leftrightarrow \bigcirc^{2j-1+2P(n)+1} C)}_{\text{content of all cells } \ne i \text{ unchanged}}$$

$$\wedge \quad \underbrace{\bigcirc^{2i-1+2P(n)+1} B}_{\text{overwrite } A \text{ by } B \text{ in cell } i}$$

$$\wedge \quad \underbrace{\bigcirc^{2i-1+2P(n)+1-2} p}_{\text{move to state } p \text{ and cursor to cell } i-1}$$

$$\varphi_\delta = \bigwedge_{q,A} \varphi_{q,A} \qquad \left[C \text{ short for } \bigvee_{r,j}(r, C, j),\ p \text{ short for } \bigvee_{D,j}(p, D, j)\right]$$

- Starting configuration
  $\varphi^w_{start} = begin \wedge \bigcirc q_0 \wedge \bigwedge_{1 \le i \le n} \bigcirc^{2i-1} A_i \wedge \bigwedge_{n < i \le P(n)} \bigcirc^{2i-1} blank$

- Accepting configuration
  $\varphi_{accept} = \Diamond \bigvee_{q \in F} q$

- Full encoding
  $\varphi_w = \varphi_{conf} \wedge \varphi^w_{start} \wedge \varphi_\delta \wedge \varphi_{accept}$
  $\Rightarrow$ Model check $\neg\varphi_w$.

PSPACE-completeness

Claim: The LTL model checking problem can be solved by a nondeterministic polynomial space-bounded algorithm

Idea: Guess, nondeterministically, an accepting run in $TS \times G_\varphi$:

$u_0 u_1 \dots u_{n-1} (v_0 v_1 \dots v_{m-1})^\omega$

where $n, m \leq |S| \cdot 2^{|\varphi|}$

- Guess n, m nondeterministically by guessing $\lceil \log(|S| \cdot 2^{|\varphi|}) \rceil = O(\log(|S|) \cdot |\varphi|)$ bits.
- Guess the sequence $u_0 u_1 \dots u_{n-1} u_n \dots u_{n+m}$ where $u_i = (s_i, B_i)$ such that
  - s_i is a successor of s_{i-1} for $i \geq 1$
  - B_i is elementary
  - $B_i \cap AP = L(s_i)$
  - $B_i \in \delta(B_{i-1}, L(s_{i-1}))$ for $i \geq 1$.
- Check if $u_n = u_{n+m}$
- Check that whenever $\varphi_1 \, \mathsf{U} \, \varphi_2 \in B_i$ for some $i \in \{n, \dots n+m-1\}$ then $\exists j \in \{n, \dots, n+m-1\}$ with $\varphi_2 \in B_j$

Required space

$n + m$ can be exponential. However, we only need:

- pair of states u_{i-1}, u_i;
- flag which $\varphi_1 \, \mathsf{U} \, \varphi_2$ have appeared in loop;
- flag which φ_2 have appeared;
- u_n

$\Rightarrow$ polynomial space

LTL satisfiability and validity checking

- Satisfiability problem: $\mathit{Words}(\varphi) \neq \varnothing$ for LTL-formula φ?
  - does there exist a transition system for which φ holds?
- Solution: construct an NBA $\mathcal{A}_\varphi$ and check for emptiness
  - nested depth-first search for checking persistence properties
- Validity problem: is $\varphi \equiv \text{true}$, i.e., $\mathit{Words}(\varphi) = \left(2^{AP}\right)^\omega$?
  - does φ hold for every transition system?
- Solution: as for satisfiability, as φ is valid iff $\neg\varphi$ is not satisfiable

runtime is exponential;
a more efficient algorithm most probably does not exist!

LTL satisfiability and validity checking

The satisfiability and validity problem for LTL are PSPACE-complete

Idea: Reduce model checking problem of φ to satisfiability problem by encoding transition system as LTL formula:

$$\psi = \psi_I \wedge \Box\, \psi_S \wedge \Box\, \psi_{AP}$$

- $\psi_I = \bigvee_{q \in I} q$
- $\psi_S = \bigwedge_{q \in S} q \rightarrow \bigcirc \bigvee_{q' \in Post(q)} q'$
- $\psi_{AP} = \bigwedge_{q \in S} q \rightarrow \bigwedge_{a \in L(q)} a \wedge \bigwedge_{a \notin L(q)} \neg a$

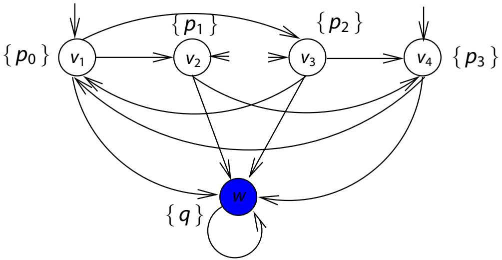
Check satisfiability of $\psi \wedge \neg\varphi$.

Model-checking LTL versus CTL

- Model-checking LTL
  - linear in the state space of the system model
  - exponential in the length of the formula
- Model-checking CTL
  - linear in the state space of the system model and the formula
- Is model-checking CTL more efficient?

Hamiltonian path problem

⇒ LTL-formulae can be exponentially shorter than their CTL-equivalent



- Existence of Hamiltonian path in LTL:
  $\bigwedge_i \big(\Diamond p_i \;\wedge\; \Box(p_i \rightarrow \bigcirc \Box \neg p_i) \big)$
- In CTL, all possible (= 4!) routes need to be encoded

Summary of LTL model checking (1)

- LTL is a logic for formalizing path-based properties
- Expansion law allows for rewriting until into local conditions and next
- LTL-formula φ can be transformed algorithmically into NBA $\mathcal{A}_\varphi$
  - this may cause an exponential blow up
  - algorithm: first construct a GNBA for φ; then transform it into an equivalent NBA
- LTL-formulae describe ω-regular LT-properties
  - but do not have the same expressivity as ω-regular languages

Summary of LTL model checking (2)

- $TS \models \varphi$ can be solved by a nested depth-first search in $TS \otimes \mathcal{A}_{\neg\varphi}$
  - time complexity of the LTL model-checking algorithm is linear in *TS* and exponential in $|\varphi|$
- Fairness assumptions can be described by LTL-formulae

  the model-checking problem for LTL with fairness is reducible to the standard LTL model-checking problem
- The LTL-model checking problem is PSPACE-complete
- Satisfiability and validity of LTL amounts to NBA emptiness-check
- The satisfiability and validity problems for LTL are PSPACE-complete

Bounded model checking

BDD vs. SAT based approaches

BDD-based approaches

- Approach used by many "industrial-strength" model checkers
- Hundreds of state variables
- Canonical representation $\Rightarrow$ BDDs often too large
- Variable order uniform along all paths, selection of good order very difficult

SAT-based approaches

- Avoid space explosion of BDDs
- Different split orders possible on different branches
- Very efficient implementations available

Bounded model checking: Basic idea

Search for counterexamples of bounded length

There exists a counterexample of length k to the invariant AG p
iff the following formula is satisfiable:

$$f_I(\vec{v}_0) \wedge f_{\rightarrow}(\vec{v}_0, \vec{v}_1) \wedge f_{\rightarrow}(\vec{v}_1, \vec{v}_2) \wedge \ldots f_{\rightarrow}(\vec{v}_{k-2}, \vec{v}_{k-1}) \wedge (\neg p_0 \vee \neg p_1 \vee \ldots \vee \neg p_{k-1})$$

Example: two-bit counter

- Initial state: $f_I = (\neg l \land \neg r)$
- Transition: $f_{\rightarrow}(l, r, l', r') = (r' \leftrightarrow \neg r) \land (l' \leftrightarrow (l \leftrightarrow \neg r))$
- Property: $\mathrm{AG}\,(\neg l \lor \neg r)$

Counterexample of length 3?

$$\underbrace{\neg l_0 \land \neg r_0}_{f_I(\vec{v}_0)} \land \underbrace{r_1 \leftrightarrow \neg r_0 \land l_1 \leftrightarrow (l_0 \leftrightarrow \neg r_0)}_{f_{\rightarrow}(\vec{v}_0,\vec{v}_1)}$$

$$\land \underbrace{r_2 \leftrightarrow \neg r_1 \land l_2 \leftrightarrow (l_1 \leftrightarrow \neg r_1)}_{f_{\rightarrow}(\vec{v}_1,\vec{v}_2)} \land (\underbrace{l_0 \land r_0}_{\neg p_0} \lor \underbrace{l_1 \land r_1}_{\neg p_1} \lor \underbrace{l_2 \land r_2}_{\neg p_2})$$

unsatisfiable $\Rightarrow$ no counterexample

Example: two-bit counter

- Initial state: $f_I = (\neg l \wedge \neg r)$
- Transition: $f_{\rightarrow}(l, r, l', r') = (r' \leftrightarrow \neg r) \;\wedge\; (l' \leftrightarrow (l \leftrightarrow \neg r))$
- Property: $\mathrm{AG}\,(\neg l \vee \neg r)$

Counterexample of length 4?

$$\underbrace{\neg l_0 \wedge \neg r_0}_{f_I(\vec{v}_0)} \wedge \underbrace{r_1 \leftrightarrow \neg r_0 \;\wedge\; l_1 \leftrightarrow (l_0 \leftrightarrow \neg r_0)}_{f_{\rightarrow}(\vec{v}_0,\vec{v}_1)} \wedge \underbrace{r_2 \leftrightarrow \neg r_1 \;\wedge\; l_2 \leftrightarrow (l_1 \leftrightarrow \neg r_1)}_{f_{\rightarrow}(\vec{v}_1,\vec{v}_2)}$$

$$\wedge \underbrace{r_3 \leftrightarrow \neg r_2 \;\wedge\; l_3 \leftrightarrow (l_2 \leftrightarrow \neg r_2)}_{f_{\rightarrow}(\vec{v}_2,\vec{v}_3)} \wedge (\underbrace{l_0 \wedge r_0}_{\neg p_0} \vee \underbrace{l_1 \wedge r_1}_{\neg p_1} \vee \underbrace{l_2 \wedge r_2}_{\neg p_2} \vee \underbrace{l_3 \wedge r_3}_{\neg p_3})$$

satisfiable $\Rightarrow$ counterexample!

SAT

- Given a propositional formula ψ,
  does there exist a variable assignment
  under which ψ evaluates to true?
- NP-complete
- In practice, tremendous progress over the last years
- Most solvers use Conjunctive Normal Form (CNF)
- Arbitrary formulas can be transformed in polynomial time into satisfiability equivalent formulas in CNF

Davis-Putnam-Logemann-Loveland (DPLL) algorithm

```
if preprocess() = CONFLICT then
        return UNSAT;
while TRUE do
        if not decide-next-branch() then
                return SAT;
        while deduce() = CONFLICT do
                blevel := analyze-conflict();
                if blevel=0 then
                        return UNSAT;
                backtrack(blevel);
        done;
done;
```

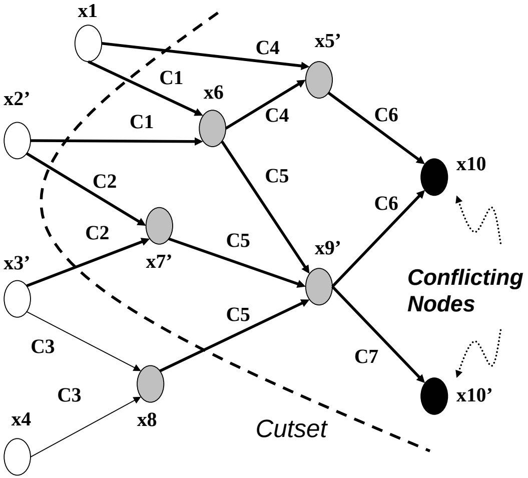
Conflict analysis using an implication graph

Clauses:
C1: x1'+ x2 + x6
C2: x2 + x3 + x7'
C3: x3 + x4'+ x8
C4: x1'+ x6'+ x5'
C5: x6'+ x7+ x8'+ x9'
C6: x5 + x9 + x10
C7: x9 + x10'

Conflict Clause C8:
x1'+ x2 + x3 + x8'

Due to conflict
(x10, x10')

Implication Graph



Prasad/Biere/Gupta: *A Survey of Recent Advances in SAT-Based Formal Verification*

Efficiency

- conflict learning: adding conflict clauses
- non-chronological backtracking
- heuristics for decisions
- efficient data structures
- incremental satisfiability