Formal Methods and Functional Programming

Model Checking

Peter Müller

Chair of Programming Methodology
ETH Zurich

LTL Model Checking Problem

Given a finite transition system TS and an LTL formula ϕ,
decide whether $t \vDash \phi$ for all $t \in \mathcal{T}(TS)$

- We need to check inclusion of traces
  - LTL formula ϕ describes a set of traces $P(\phi)$
  - We need to determine whether or not $\mathcal{T}(TS) \subseteq P(\phi)$
- Naïvely searching all traces is not an option (infinite length)

Checking Regular Safety Properties

- A safety property is regular if its bad prefixes are described by a regular language over the alphabet $\mathcal{P}(AP)$
- Every invariant over AP is a regular safety property
  - For the property defined by $\Box p$, all bad prefixes start with $S^* T$ where S describes any subset of $\mathcal{P}(AP)$ that contains p, and T any subset that does not contain p
  - For example, bad prefixes for $\Box open$ are described by $(\{open\} \mid \{open, closed\})^* (\{\} \mid \{closed\})$
- Non-regular safety properties also exist
  - Vending machine: at least as many coins inserted as drinks dispensed
  - Bad prefixes: regular languages "cannot count"

Checking Regular Safety Properties: Approach

- Look at all finite prefixes $\mathcal{T}_{fin}(TS)$ of the traces $\mathcal{T}(TS)$ of a transition system TS
- Check whether $\mathcal{T}_{fin}(TS)$ contains a bad prefix

Checking Regular Safety Properties: Approach

- Look at all finite prefixes $\mathcal{T}_{fin}(TS)$ of the traces $\mathcal{T}(TS)$ of a transition system TS
- Check whether $\mathcal{T}_{fin}(TS)$ contains a bad prefix

1. Describe finite prefixes $\mathcal{T}_{fin}(TS)$ by finite automaton $\mathcal{FA}_{TS}$

Checking Regular Safety Properties: Approach

- Look at all finite prefixes $\mathcal{T}_{fin}(TS)$ of the traces $\mathcal{T}(TS)$ of a transition system TS
- Check whether $\mathcal{T}_{fin}(TS)$ contains a bad prefix

1. Describe finite prefixes $\mathcal{T}_{fin}(TS)$ by finite automaton $\mathcal{FA}_{TS}$
2. Describe bad prefixes of regular safety property P by finite automaton $\mathcal{FA}_{\bar{P}}$

Checking Regular Safety Properties: Approach

- Look at all finite prefixes $\mathcal{T}_{fin}(TS)$ of the traces $\mathcal{T}(TS)$ of a transition system TS
- Check whether $\mathcal{T}_{fin}(TS)$ contains a bad prefix

1. Describe finite prefixes $\mathcal{T}_{fin}(TS)$ by finite automaton $\mathcal{FA}_{TS}$
2. Describe bad prefixes of regular safety property P by finite automaton $\mathcal{FA}_{\bar{P}}$
3. Construct finite automaton for product of $\mathcal{FA}_{TS}$ and $\mathcal{FA}_{\bar{P}}$

Checking Regular Safety Properties: Approach

- Look at all finite prefixes $\mathcal{T}_{fin}(TS)$ of the traces $\mathcal{T}(TS)$ of a transition system TS
- Check whether $\mathcal{T}_{fin}(TS)$ contains a bad prefix

1. Describe finite prefixes $\mathcal{T}_{fin}(TS)$ by finite automaton $\mathcal{FA}_{TS}$
2. Describe bad prefixes of regular safety property P by finite automaton $\mathcal{FA}_{\bar{P}}$
3. Construct finite automaton for product of $\mathcal{FA}_{TS}$ and $\mathcal{FA}_{\bar{P}}$
4. Check if the resulting automaton has any reachable accepting states
   - If not, the property P is never violated by computations of TS
   - If yes, the property P is violated
     Each word in the accepted language of the product automaton is a counterexample
     (i.e., a bad prefix of P that is a prefix of a computation of TS)

Reminder: Finite Automata

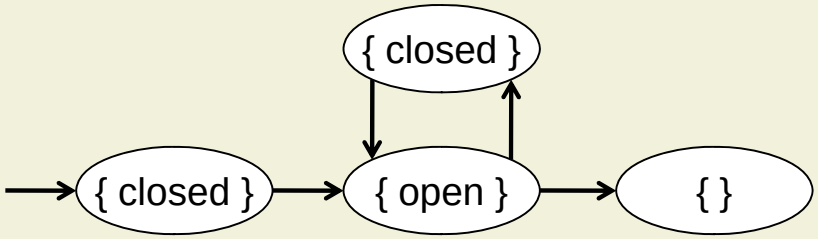
- A finite automaton (FA) is a tuple $(Q, \Sigma, Q, \delta, q_0, F)$
  - Q: a finite set of states
  - Σ: a finite alphabet
  - δ: a transition relation, $\delta \subseteq Q \times \Sigma \times Q$
  - q_0: an initial state
  - $F \subseteq Q$: a set of accepting states

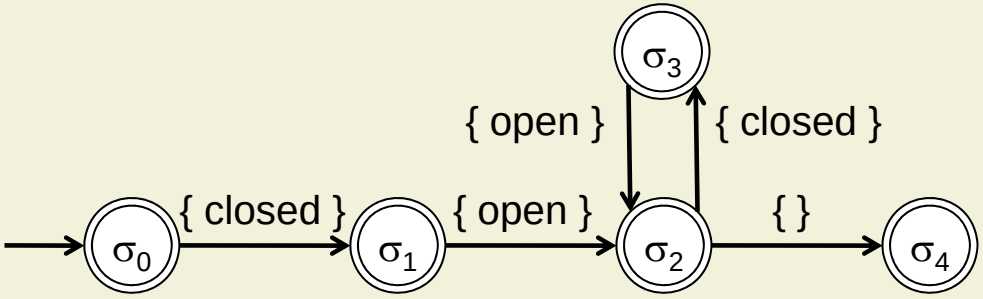
Step 1: Finite Automaton for Finite Prefixes

- Given a transition system $TS = (\Gamma, \sigma_I, \rightarrow)$, propositions AP, and labeling function L

- The automaton $\mathcal{FA}_{TS} = (Q, \Sigma, \delta, q_0, F)$ accepts $\mathcal{T}_{fin}(TS)$
  - $Q = \Gamma \cup \{\sigma_0\}$, where $\sigma_0 \notin \Gamma$
  - $\Sigma = \mathcal{P}(AP)$
  - $\delta = \{(\sigma, p, \sigma') \mid \sigma \rightarrow \sigma' \text{ and } p \in L(\sigma')\} \cup \{(\sigma_o, p, \sigma_I) \mid p \in L(\sigma_I)\}$
  - $q_0 = \sigma_0$
  - $F = Q$ (accept any prefix of a computation)

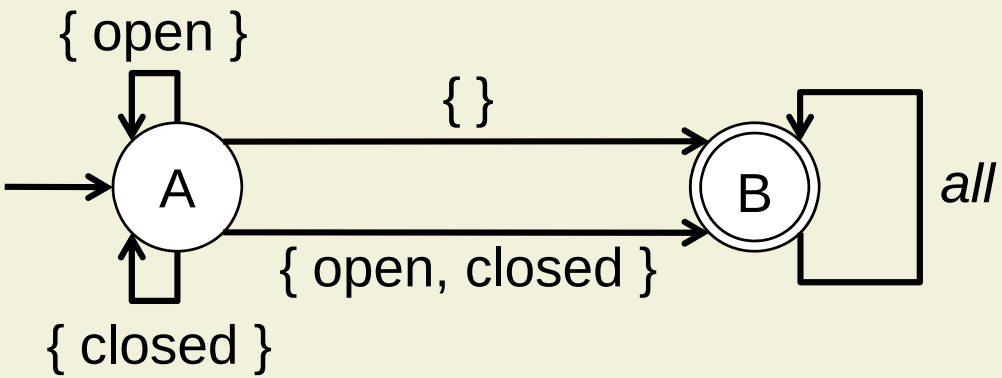
- Example: `o:=o+1; while * do o:=o-1; o:=o+1 end; o:=o+1`





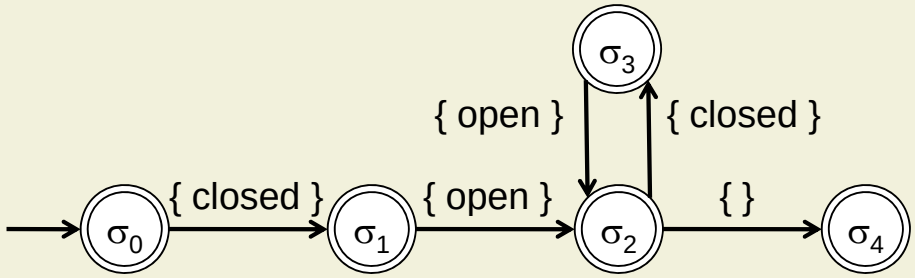
Step 2: Finite Automaton for Bad Prefixes

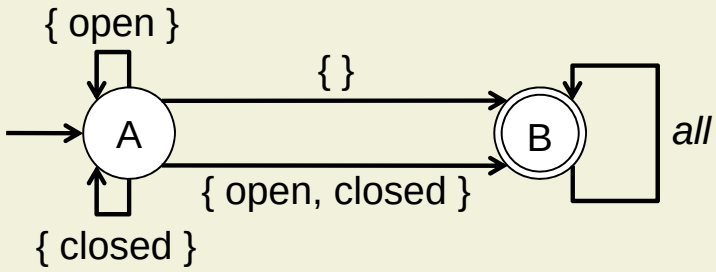
- By definition, bad prefixes are described by a regular language
- Apply standard construction to obtain FA $\mathcal{FA}_{\bar{P}}$ from regular expression
- Example: $\Box((open \vee closed) \wedge \neq (open \wedge closed))$
  - Bad prefixes start with
    $(\{open\} \mid \{closed\})^*(\{\} \mid \{open, closed\})$

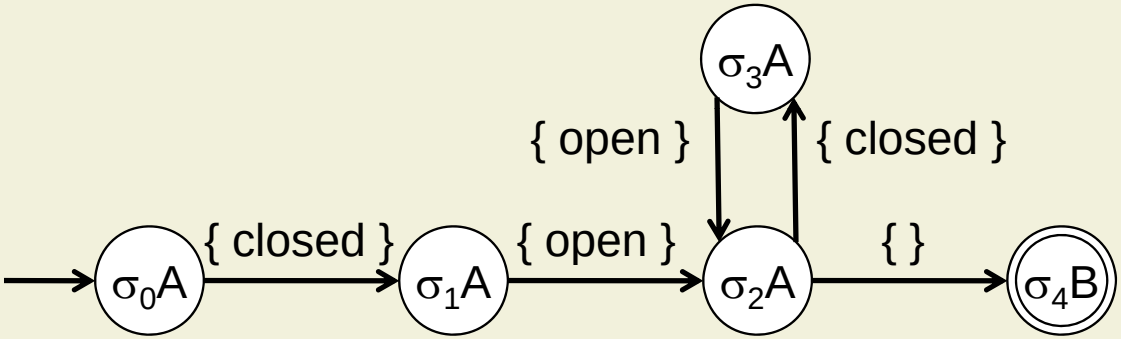


Step 3: Finite Automaton for Product

- Construct FA $\mathcal{FA}_{TS \cap \bar{P}}$ that accepts the intersection of the languages accepted by $\mathcal{FA}_{TS}$ and $\mathcal{FA}_{\bar{P}}$
- Apply standard construction for product of two FA
- Example





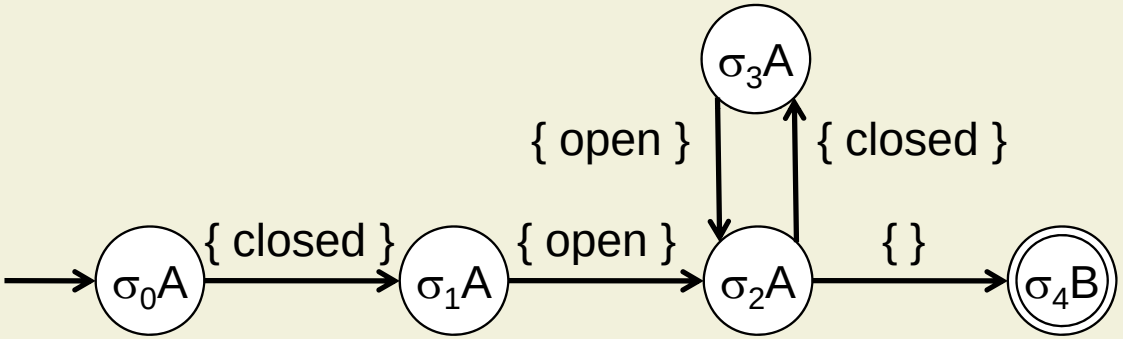


Step 4: Check Emptiness

- If $\mathcal{FA}_{TS \cap \bar{P}}$ accepts a word w then
  - $w \in \mathcal{T}_{fin}(TS)$ because it is accepted by $\mathcal{FA}_{TS}$ and
  - w is a bad prefix because it is accepted by $\mathcal{FA}_{\bar{P}}$
  - Therefore, P is not satisfied, and w is a counterexample

- Apply standard algorithm to check emptiness of FA

- Example



  - Accepts $\{closed\}\{open\}(\{closed\} \mid \{open\})^*\{\}$
  - Smallest counterexample: $\{closed\}\{open\}\{\}$
  - Counterexample can be mapped back to transition system

Büchi Automata

- Büchi automata are similar to finite automata, but accept infinite words
- A Büchi automaton (BA) is a tuple $(Q, \Sigma, Q, \delta, q_0, F)$
  - Q: a finite set of states
  - Σ: a finite alphabet
  - δ: a transition relation, $\delta \subseteq Q \times \Sigma \times Q$
  - q_0: an initial state
  - $F \subseteq Q$: a set of accepting states
- A run of a BA accepts its input if it passes infinitely often through an accepting state
- Büchi automata enjoy many of the properties of finite automata
  - We can construct the product of two BA
  - Emptiness is decidable

LTL Model Checking: Approach

1. Describe traces $\mathcal{T}(TS)$ by Büchi automaton $\mathcal{BA}_{TS}$
   - Construction is analogous to $\mathcal{FA}_{TS}$

LTL Model Checking: Approach

1. Describe traces $\mathcal{T}(TS)$ by Büchi automaton $\mathcal{BA}_{TS}$
   - Construction is analogous to $\mathcal{FA}_{TS}$

2. For an LTL formula ϕ, construct Büchi automaton $\mathcal{BA}_{\neg\phi}$ that accepts the traces characterized by $\neg\phi$ (bad traces)
   - We omit the details here

LTL Model Checking: Approach

1. Describe traces $\mathcal{T}(TS)$ by Büchi automaton $\mathcal{BA}_{TS}$
   - Construction is analogous to $\mathcal{FA}_{TS}$

2. For an LTL formula ϕ, construct Büchi automaton $\mathcal{BA}_{\neg\phi}$ that accepts the traces characterized by $\neg\phi$ (bad traces)
   - We omit the details here

3. Construct BA for product of $\mathcal{BA}_{TS}$ and $\mathcal{BA}_{\neg\phi}$

LTL Model Checking: Approach

1. Describe traces $\mathcal{T}(TS)$ by Büchi automaton $\mathcal{BA}_{TS}$
   - Construction is analogous to $\mathcal{FA}_{TS}$

2. For an LTL formula ϕ, construct Büchi automaton $\mathcal{BA}_{\neg\phi}$ that accepts the traces characterized by $\neg\phi$ (bad traces)
   - We omit the details here

3. Construct BA for product of $\mathcal{BA}_{TS}$ and $\mathcal{BA}_{\neg\phi}$

4. Check whether intersection is empty
   - If intersection is non-empty, property ϕ is violated
   - Each word in the intersection is a counterexample

Complexity Results

For a finite transition system TS and an LTL formula ϕ, the model checking problem $TS \models \phi$ is solvable in $\mathcal{O}(\mid TS \mid \times 2^{\mid \phi \mid})$

- $\mid TS \mid$ is the size of the transition system (which grows exponentially in the number of variables, processes, and channels)
- $\mid \phi \mid$ is the size of ϕ; exponential complexity comes from the construction of $\mathcal{BA}_{\neg\phi}$

Advanced Model Checking Techniques

- On-the-fly model checking
  - Often violation of a property can be detected without checking all possible states or traces (for instance, $\Box p$)
  - Generate transition system and check property step-by-step
  - Implemented in Spin

- Partial order reduction
  - Remove redundancy from different interleavings of concurrent executions
  - Code segments that operate only on local state are not affected by interleaving
  - Implemented in Spin

Advanced Model Checking Techniques (cont'd)

- Bounded model checking
  - Check only prefixes of traces up to a certain length
  - Closer to testing than verification
  - Very effective in practice

- Symbolic model checking
  - Uses sets of states rather than individual states
  - Sets of states are represented through boolean functions
  - Very efficient data structure: binary decision diagram (BDDs)
  - Typically used to check branching-time properties
  - Can deal with larger models

Conclusions

- Variety of approaches
  - Best method depends on application area
- Tool support is essential
  - Proofs are tedious and error-prone
  - Some tools have reached maturity for industrial applications