More Explicit State Model Checking

Computer Science and Artificial Intelligence Laboratory
MIT
Armando Solar-Lezama

Nov 27, 2013
Example

G rec $\rightarrow$ F ack

$\Sigma$-rec

ack

rec

$\Sigma$-ack

S1

S2

S3

rec

ack
Buchi Automata

A Buchi Automaton is a 4-tuple $\langle \Sigma, S, I, \delta, F \rangle$

- $\Sigma$ is an alphabet
- $S$ is a finite set of states
- $I \subseteq S$ is a set of initial states
- $\delta \subseteq S \times \Sigma \times S$ is a transition relation
- $F \subseteq S$ is a set of accepting states

Accept if $F$ is visited infinitely often.
A Buchi Automaton is a 4-tuple

$\langle \Sigma, S, I, \delta, \{F_1, ... F_n\} \rangle$

- $\Sigma$ is an alphabet
- $S$ is a finite set of states
- $I \subseteq S$ is a set of initial states
- $\delta \subseteq S \times \Sigma \times S$ is a transition relation
- $\{F_1, ... F_n\} \subseteq \mathcal{P}(S)$ is a set of accepting states

Accept if each $F_i$ is visited infinitely often
Given \( \langle \Sigma, S, I, \delta, \{F_1, \ldots, F_n\} \rangle \)

The NBA is defined as follows:

\[ \langle \Sigma, S', I', \delta', F' \rangle \]

- \( S' = S \times \{1, \ldots, n\} \)
- \( I' = I \times \{1\} \)
- \( \delta' = \{(a, i), e, (b, j)\} \mid a \in F_i \Rightarrow j = (i + 1) \mod n \)
- \( a \notin F_i \Rightarrow j = i \)
- \( F' = F_1 \times \{1\} \)

Why does this work?
Closure of an LTL formula

Given a formula $\varphi$, $\text{closure}(\varphi)$ is the set of all subformulas and their negation.

Ex:

- $\text{closure}(p \land (q U r))$

- $\text{closure}(G(p \Rightarrow F q)) = \text{closure}(\neg t U (\neg (\neg p \lor t U q)) )$
  $\text{closure}(\neg t U (p \land \neg (t U q)))$
Elementary Sets of Formulae

A set $B \subseteq \text{closure}(\varphi)$ is elementary iff:

- It is consistent with respect to propositional logic:
  - $f \land g \in B \Rightarrow f \in B \land g \in B$
  - $f \lor g \in B \Rightarrow f \in B \lor g \in B$
  - $\neg f \in B \Rightarrow f \notin B$

- It is locally consistent with respect to the until operator
  - this is just an application of the previous requirement, assuming the equality $p \mathbin{U} q = q \lor (p \land X(p \mathbin{U} q))$
  - $p \mathbin{U} q \in B \Rightarrow q \in B \lor p \in B$

- It is maximal
  - If $E \subset B \subseteq \text{closure}(\varphi)$ then $E$ and $B$ can’t both be elementary sets
  - maximality implies that $f \in B$ iff $\neg f \notin B$
LTL to GBA algorithm

Given $\varphi$ Consider the automata

$$\langle \Sigma, S, I, \delta, F \rangle$$

- $\Sigma = 2^{AP}$
- $S = \{B \mid B \subseteq \text{closure}(\varphi) \land B \text{ is elementary}\}$
- $I = \{B \in S \mid \varphi \in B\}$
- $F = \{F_{\varphi_1} \cup \varphi_2 \mid \varphi_1 \cup \varphi_2 \in \text{closure}(\varphi)\}$ and
  $$F_{\varphi_1} \cup \varphi_2 = \{B \in S \mid \varphi_1 U \varphi_2 \not\in B \text{ or } \varphi_2 \in B\}$$

- $\delta(B, A) = B'$ if $A \subseteq B \cap AP$

where $B'$ is defined as follows:

- for every $Xp \in \text{closure}(\varphi) : Xp \in B \iff p \in B'$
- for every $pUq \in \text{closure}(\varphi) : pUq \in B \iff q \in B \lor (p \in B \land pUq \in B')$
OPTIMIZATIONS
Graph Construction and Tracking

Construct graphs incrementally
- Compute products on the fly
- especially important for concurrent systems
  • the Kripke structure is itself a product.

Tracking visited nodes
- What happens if we forget we visited something?
- What happens if we think we visited something we didn’t?
Example

while(*){
    pc=0  if(p=0){
        p:=1;
    }
    pc=1  p:=1;
    pc=2  if(p=1){
        if(g=free){
            if(g=free){
                g:=id;
                p:=2;
            }
            g:=id;
            p:=2;
        }
        pc=3  g:=free
        pc=4  if(p=2){
            pc=6  if(p=2){
                pc=7  p:=3; g:=free
                pc=8  if(p=3){
                    pc=9  p:=0;
                }
                pc=0  if(p=0){
                    pc=1  p:=1;
                }
            }
        }
    }
}

H train

V train
Optimizations: Partial Order Reduction

Example

```
while(*){
    pc=0
    if(p=0){
        p:=1;
    }
    if(p=1){
        if(g=free){
            g:=id;
            p:=2;
        }
    }
    if(p=2){
        p:=3;
        g:=free
    }
    if(p=3){
        p:=0;
    }
}
```

```
while(*){
    pc=0
    if(p=0){
        p:=1;
    }
    if(p=1){
        if(g=free){
            g:=id;
            p:=2;
        }
    }
    if(p=2){
        p:=3;
        g:=free
    }
    if(p=3){
        p:=0;
    }
}
```

H train
V train
Partial Order Reduction

Key idea:
- The order of independent actions on different threads does not matter
- Note: what is considered independent depends on the property
Ample set

On state s1, the transitions to s2 and s3 are both enabled.
- enabled(s1)

We only want to explore a subset of the enabled set
- ample(s1) ⊆ enabled(s1)
Ample set

We have 3 goals in computing *ample(s)*
- Using ample instead of enabled should give us a much smaller graph
- Using ample instead of enabled should still allow us to find what we are looking for
- Computing ample should be easy
Independence and Invisibility

Independence:
- Actions $a$ and $b$ are independent iff:
  - $a$ does not disable $b$ and vice-versa
  - Commutativity: $a(b(s)) = b(a(s))$

Invisibility:
- $a$ and $b$ should not affect the values of any relevant property
Ample is computed heuristically

Computing it precisely is too hard, but we can find actions that are definitely not in ample(s) and can therefore be ignored.

What we need to consider:
- Actions that share variables with the property
- If two actions share variables, they are dependent
- If two actions appear in the same thread they are dependent
Approximations

Suppose you want to ensure that
- $G \text{req} \rightarrow F \text{ack}$

But, your checker only supports CTL
- If you want to verify correctness, you can overapproximate
- $AG (\text{req} \rightarrow AF \text{ack})$