Computer-Aided Program Design

Spring 2015, Rice University

Unit 3

Swarat Chaudhuri

February 5, 2015

Temporal logic

- Propositional logic is a good language for describing properties of *program states*.
- However, programs have *dynamics*: a program's state evolves during its execution.
- *Temporal* logics are useful for reasoning about this dynamics.
  - Long history: goes back to Greek philosophers
  - Introduced to computer science by Pnueli and Manna.

(Propositional) Linear Temporal Logic: Syntax

- Let *Prop* be a set of *propositional variables*. A formula φ in (propositional) Linear Temporal Logic (LTL) has the form

$$\varphi ::= p \mid \top \mid \bot \mid \neg\varphi_1 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \mathbf{X}\ \varphi_1 \mid \varphi_1\, \mathbf{U}\, \varphi_2$$

  where $p \in \mathit{Prop}$.
- In the above, φ_1 and φ_2 are *subformulas* of φ.
- A *state formula* is a formula of the form p or $\neg p$, where $p \in \mathit{Prop}$.

[http://www.cmi.ac.in/∼madhavan/papers/pdf/isical97.pdf.]

LTL: Semantics

Interpretations are infinite words over 2^{Prop}:

$w = \{P, Q\}.\{Q\}.\{Q, R\}.\cdots$

$$
\begin{array}{llll}
w, m \models \varphi & \text{if } \varphi \text{ evaluates to} & \text{true} & \text{on } w \text{ at time point } m \\
w, m \not\models \varphi & & \text{false} &
\end{array}
$$

LTL: Semantics

Interpretations are infinite words over 2^{Prop}:

$w = \{P, Q\}.\{Q\}.\{Q, R\}.\cdots$

$w, m \models \varphi$	if φ evaluates to true	on w at time point m
$w, m \not\models \varphi$	false	

Inductive definition of semantics:

$$
\begin{array}{rlrl}
w, m & \models \top & & w, m \not\models \bot \\
w, m & \models P & \text{iff} & P \in w_m \\
w, m & \models \neg\varphi & \text{iff} & w, m \not\models \varphi \\
w, m & \models \varphi_1 \wedge \varphi_2 & \text{iff} & w, m \models \varphi_1 \text{ and } w, m \models \varphi_2 \\
w, m & \models \varphi_1 \vee \varphi_2 & \text{iff} & w, m \models \varphi_1 \text{ or } w, m \models \varphi_2 \\
w, m & \models \mathbf{X}\,\varphi_1 & \text{iff} & w, m+1 \models \varphi_1 \\
w, m & \models (\varphi_1 \,\mathbf{U}\, \varphi_2) & \text{iff} & \exists m' \geq m : \ w, m' \not\models \varphi_2 \text{ and} \\
& & & \forall m \leq n < m' : \ w, n \models \varphi_1.
\end{array}
$$

Exercises

Write the following properties in LTL:

- Eventually, P will hold. This is abbreviated as $\mathbf{F}\, P$ or $\Diamond\ P$.
- P *always* holds. This is abbreviated as $\mathbf{G}\, P$ or $\Box\ P$.

Exercises

Write the following properties in LTL:

- Eventually, P will hold. This is abbreviated as $\mathbf{F}\, P$ or $\Diamond\ P$.
- P *always* holds. This is abbreviated as $\mathbf{G}\, P$ or $\Box\ P$.
- After a certain "stable point", P holds forever.

Exercises

Write the following properties in LTL:

- Eventually, P will hold. This is abbreviated as $\mathbf{F}\, P$ or $\Diamond\ P$.
- P *always* holds. This is abbreviated as $\mathbf{G}\, P$ or $\Box\ P$.
- After a certain "stable point", P holds forever.
- P holds infinitely often in the trace.

Exercises

Write the following properties in LTL:

- Eventually, P will hold. This is abbreviated as $\mathbf{F}\, P$ or $\Diamond\, P$.
- P *always* holds. This is abbreviated as $\mathbf{G}\, P$ or $\Box\, P$.
- After a certain "stable point", P holds forever.
- P holds infinitely often in the trace.
- Consider a system that schedules access to a shared resource among k competing processes named $1, \ldots, k$. Express:
  - Mutual exclusion
  - Processes are scheduled "fairly"
  - When a process requests the resource, it stays in a requesting state until the request is granted.

Exercises

Write in LTL:

- If A occurs at least twice, then A occurs infinitely often.
- A holds at all states s_{3k} and does not hold at all states s_{3k+1}, s_{3k+2}, where $k = 0, 1, \ldots$.

Exercises

What do the following properties mean?

- $\mathbf{F}\,\mathbf{G}\,A$

Exercises

What do the following properties mean?

- $\mathbf{F}\,\mathbf{G}\,A$
- $\mathbf{F}\,\mathbf{G}\,(A \rightarrow \mathbf{X}\,A)$

Exercises

What do the following properties mean?

- $\mathbf{F}\,\mathbf{G}\,A$
- $\mathbf{F}\,\mathbf{G}\,(A \rightarrow \mathbf{X}\,A)$
- $A\,\mathbf{U}\,\neg A$

LTL verification

- Given: Symbolic transition system $\mathcal{M}$, LTL property φ.
- Question: Do all paths in $\mathcal{M}$ starting from an initial state satisfy φ?

To start with, let us assume with a transition system that is represented explicitly:

$$(\mathit{States}, \mathit{InitStates}, \dashrightarrow, \lambda)$$

- States is a finite set of states
- $\mathit{InitStates} \subseteq \mathit{States}$ is a finite set of initial states
- $\dashrightarrow \subseteq \mathit{States} \times \mathit{States}$ is a transition relation
- $\lambda : \mathit{States} \rightarrow 2^{\mathit{Prop}}$ is a labeling of states with atomic propositions.

Key: relationship between LTL and automata

- A *Büchi automaton* $\mathcal{A}$ is a finite automaton that runs on *infinite words*
- Definition:

$$\mathcal{A} = (S, \Sigma, S_{in}, \hookrightarrow, G)$$

  - S is a set of states
  - Σ is an input alphabet
  - $S_{in} \subseteq S$ is a set of initial states
  - $\hookrightarrow \subseteq S \times \Sigma \times S$ is a transition relation
  - G is a set of *accepting* states.

Semantics of Büchi automata

- Let $w = w_0 w_1 w_2 \ldots$ be an infinite word over Σ.
- *Run* of $\mathcal{A}$ on w: infinite word $\rho = s_0 s_1 s_2 \ldots$ over S such that
  - $s_0 \in S_{in}$
  - for all $i \geq 0$, $(s_i, w_i, s_{i+1}) \in \hookrightarrow$.
- The run ρ is accepting iff an accepting state appears infinitely often in ρ
- $\mathcal{A}$ *accepts* w iff $\mathcal{A}$ has an accepting run on w.

Büchi automata

- Construct an automaton with input alphabet $\{a, b\}$ that accepts:
  - All words where a occurs infinitely often

Büchi automata

- Construct an automaton with input alphabet $\{a, b\}$ that accepts:
  - All words where a occurs infinitely often
  - All words where a occurs only finitely often

Büchi automata

- Construct an automaton with input alphabet $\{a, b\}$ that accepts:
  - All words where a occurs infinitely often
  - All words where a occurs only finitely often
  - All words with an infinite suffix consisting only of b's.

Büchi automata

- Construct an automaton with input alphabet $\{a, b\}$ that accepts:
  - All words where a occurs infinitely often
  - All words where a occurs only finitely often
  - All words with an infinite suffix consisting only of b's.
- Compute a Buchi automaton that accepts the intersection of the languages of two given Buchi automata.

LTL to Büchi automata

Every LTL formula φ can be converted to a Büchi automaton $\mathcal{A}_\varphi$ over the input alphabet 2^{Prop} such that for any word w over 2^{Prop}, $w \models \varphi$ iff $\mathcal{A}_\varphi$ accepts w.

See construction in chapter `http://www.cmi.ac.in/~madhavan/papers/pdf/isical97.pdf`.

Before we go there: algorithms for Büchi automata

- Let $\mathcal{L}(\mathcal{A})$, the *language of* $\mathcal{A}$, be the set of words accepted by $\mathcal{A}$.
- Given two Buchi automata $\mathcal{A}_1$ and $\mathcal{A}_2$, construct an automaton that accepts
  - $\mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$

Before we go there: algorithms for Büchi automata

- Let $\mathcal{L}(\mathcal{A})$, the *language of* $\mathcal{A}$, be the set of words accepted by $\mathcal{A}$.
- Given two Buchi automata $\mathcal{A}_1$ and $\mathcal{A}_2$, construct an automaton that accepts
  - $\mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$
  - $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$.

Before we go there: algorithms for Büchi automata

- Let $\mathcal{L}(\mathcal{A})$, the *language of* $\mathcal{A}$, be the set of words accepted by $\mathcal{A}$.
- Given two Buchi automata $\mathcal{A}_1$ and $\mathcal{A}_2$, construct an automaton that accepts
  - $\mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$
  - $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$.
- Give an algorithm to check, for given $\mathcal{A}$, if $\mathcal{L}(\mathcal{A})$ is empty.

Generalized Büchi automata

- Büchi automata with k sets of accepting states.

$$\mathcal{A}^{\#} = (S, \Sigma, S_{in}, \hookrightarrow, \langle G_1, \ldots, G_k \rangle)$$

- A run is accepting iff it visits *every* set $G_1, \ldots, G_k$ infinitely often.

Generalized Büchi automata

- Büchi automata with k sets of accepting states.

$$\mathcal{A}^{\#} = (S, \Sigma, S_{in}, \hookrightarrow, \langle G_1, \ldots, G_k \rangle)$$

- A run is accepting iff it visits *every* set $G_1, \ldots, G_k$ infinitely often.
- Let $\mathcal{A}_i$ be $(S, \Sigma, S_{in}, \hookrightarrow, G_i)$. It is easy to see that

$$\mathcal{L}(\mathcal{A}^{\#}) = \bigcap_i \mathcal{L}(\mathcal{A}_i)$$

.

Generalized Büchi automata

- Büchi automata with k sets of accepting states.

$$\mathcal{A}^{\#} = (S, \Sigma, S_{in}, \hookrightarrow, \langle G_1, \ldots, G_k \rangle)$$

- A run is accepting iff it visits *every* set $G_1, \ldots, G_k$ infinitely often.
- Let $\mathcal{A}_i$ be $(S, \Sigma, S_{in}, \hookrightarrow, G_i)$. It is easy to see that

$$\mathcal{L}(\mathcal{A}^{\#}) = \bigcap_i \mathcal{L}(\mathcal{A}_i)$$

.

- Question: what's the complexity of converting a Generalized Buchi automaton to a regular Buchi automaton? (Hint: use a variant of the "round-robin" argument used for intersection.)

LTL to Büchi: steps

- Fischer-Ladner closure
- Atoms: construction of automaton states
- Construction of generalized Büchi automaton
- Conversion into standard Büchi automaton (optional).

LTL to Büchi: Fischer-Ladner closure

Suppose φ is the formula that we want to convert to a Buchi automaton. The (Fischer-Ladner) closure FL_φ of φ is defined as the least set of formulas such that:

- If $\varphi \in FL_\varphi$
- If $\psi \in FL_\varphi$ then $\neg\psi \in FL_\varphi$ (we identify $\neg\neg\psi$ with ψ)
- If $\psi_1 \wedge \psi_2 \in FL_\varphi$ then $\psi_1, \psi_2 \in FL_\varphi$
- If $\psi_1 \vee \psi_2 \in FL_\varphi$ then $\psi_1, \psi_2 \in FL_\varphi$
- If $\mathbf{X}\,\psi_1 \in FL_\varphi$ then $\psi_1 \in FL_\varphi$
- $\psi_1 \,\mathbf{U}\, \psi_2 \in FL_\varphi$ then $\psi_1, \psi_2, \mathbf{X}\,(\psi_1 \,\mathbf{U}\, \psi_2) \in FL_\varphi$.

Atoms

A *atom* is a set $S \subseteq FL_\varphi$ such that:

- $\psi \in S$ iff $\neg\psi \notin S$
- $\psi_1 \wedge \psi_2 \in S$ iff $\psi_1 \in S$ and $\psi_2 \in S$
- $\psi_1 \vee \psi_2 \in S$ iff $\psi_1 \in S$ or $\psi_2 \in S$
- $\psi_1 \,\mathbf{U}\, \psi_2 \in S$ iff either $\psi_2 \in S$ or $\psi_1, \mathbf{X}\,(\psi_1 \,\mathbf{U}\, \psi_2) \in S$

Let the set of all atoms constructed this way be called $Atom_\varphi$.

Compilation to generalized Buchi automaton

- Set of states: $Atom_{\varphi}$

Compilation to generalized Buchi automaton

- Set of states: $Atom_\varphi$
- Initial states: any state that contains φ

Compilation to generalized Buchi automaton

- Set of states: $Atom_\varphi$
- Initial states: any state that contains φ
- Transitions: triples (S_1, P, S_2) where:
  - $\mathbf{X}\,\psi \in S_1$ and $\psi \in S_2$ for some ψ
  - $\neg\mathbf{X}\,\psi \in S_1$ and $\neg\psi \in S_2$ for some ψ
  - P is the set of atomic propositions in S_1.

Compilation to generalized Buchi automaton

- Set of states: $Atom_\varphi$
- Initial states: any state that contains φ
- Transitions: triples (S_1, P, S_2) where:
  - $\mathbf{X}\,\psi \in S_1$ and $\psi \in S_2$ for some ψ
  - $\neg\mathbf{X}\,\psi \in S_1$ and $\neg\psi \in S_2$ for some ψ
  - P is the set of atomic propositions in S_1.
- Accepting states: For each formula $\psi_1 \,\mathbf{U}\, \psi_2 \in FL_\varphi$, define a set of accepting states G_i, consisting of all states S such that:
  - either $\psi_2 \in S$
  - or $\psi_1 \,\mathbf{U}\, \psi_2 \notin S$.

Compilation to generalized Buchi automaton

- <u>Set of states:</u> $Atom_\varphi$
- <u>Initial states:</u> any state that contains φ
- <u>Transitions:</u> triples (S_1, P, S_2) where:
  - $\mathbf{X}\,\psi \in S_1$ and $\psi \in S_2$ for some ψ
  - $\neg\mathbf{X}\,\psi \in S_1$ and $\neg\psi \in S_2$ for some ψ
  - P is the set of atomic propositions in S_1.
- <u>Accepting states:</u> For each formula $\psi_1\,\mathbf{U}\,\psi_2 \in FL_\varphi$, define a set of accepting states G_i, consisting of all states S such that:
  - either $\psi_2 \in S$
  - or $\psi_1\,\mathbf{U}\,\psi_2 \notin S$.

- You can go from generalized Buchi automaton to Buchi automaton by a variant of the intersection construction.
- For verification, you can work on generalized Buchi automaton directly.

Automata-theoretic verification for LTL

- Given:
  1. (Explicit) transition system $\mathcal{M} = (\mathit{States}, \mathit{InitStates}, \dashrightarrow, \lambda)$
  2. LTL formula φ.

  Each execution of $\mathcal{M}$ is an interpretation for φ. Let the set of all such interpretations be $\mathcal{L}(\mathcal{M})$.
- Question: Verify that all executions of $\mathcal{M}$ satisfy φ.

Automata-theoretic verification for LTL

- Given:
  1. (Explicit) transition system $\mathcal{M} = (\mathit{States}, \mathit{InitStates}, \dashrightarrow, \lambda)$
  2. LTL formula φ.

  Each execution of $\mathcal{M}$ is an interpretation for φ. Let the set of all such interpretations be $\mathcal{L}(\mathcal{M})$.
- Question: Verify that all executions of $\mathcal{M}$ satisfy φ.

Solution:

1. Convert $\neg\varphi$ into a (generalized) Buchi automaton $\overline{\mathcal{A}}$.
2. Question: $\mathcal{L}(\mathcal{M}) \cap \mathcal{L}(\overline{\mathcal{A}}) = \emptyset$?

Verification using SPIN

- SPIN: system for LTL verification of finite-state systems: `http://www.spinroot.com`.

Verification using SPIN

- SPIN: system for LTL verification of finite-state systems: `http://www.spinroot.com`.
- Formula to Buchi automaton: `spin -f ‘‘[]p’’`
- Verification:

```
$ spin -a zune.pml
$ cc pan.c
$ a.out -a
```

- Try out the examples in the `Tests` folder, in particular Peterson’s protocol and the Zune bug.

SPIN internals

- Essence of algorithm for Buchi emptiness: search for a cycle that is: (a) reachable from an initial state; and (b) contains an accepting state.
- Algorithm: Nested DFS, implemented in SPIN system.
- <u>Question:</u> What is the naive DFS algorithm for LTL verification?

Avoiding the quadratic complexity

- Suppose you have nodes u and v, and you know:
  - u is an ancestor of v in a DFS tree
  - You know that v is not in a cycle
- Theorem: No cycle containing u contains nodes reachable from v.
- In other words, you can ignore states reached from v while doing the nested search from u.

Nested DFS

```
DFS(s):
  if error then report error
  add <s,0> to Visited
  for t ∈ Next(s):
    if <t,0> ∉ Visited:
      DFS(t)
  if accepting(s):
     seed = s; NDFS(s)

NDFS(s):
  add <s,1> to Visited
  for all t in Next(s):
     if <t,1> ∉ Visited:
       NDFS(t)
     else if t == seed:
       report cycle
```

Question for discussion

- Do we really require a finite state system to apply nested DFS?

Question for discussion

- Do we really require a finite state system to apply nested DFS?
- What happens when we go from explicit transition systems to symbolic transition systems?

SAT-based bounded model checking for LTL

- Earlier, we studied bounded verification for assertions.
- In the language of LTL, they are *safety properties* of the form $\mathbf{G}\, p$.
- What about general LTL properties?

SAT-based bounded model checking for LTL

- Earlier, we studied bounded verification for assertions.
- In the language of LTL, they are *safety properties* of the form **G** p.
- What about general LTL properties?
  - Idea: search for "lassos" in a bounded way:
  - For example, to find if the property **F** p is violated in k steps or less, check satisfiability of

$$B(k) = In(X_0) \wedge \bigwedge_{i=0}^{k-1} T(X_i, X_{i+1}) \wedge \bigwedge_{i=0}^{k} \neg P(X_i) \wedge \bigvee_{i=0}^{k} (X_i = X_k)$$