CIS 842: Specification and Verification of Reactive Systems

Lecture Specifications: LTL Model Checking

Objectives

- To understand Buchi automata and the relationship to LTL
- To understand how Buchi acceptance search enables a general LTL model checking algorithm

Safety Checking

For safety properties we automated the "instrumentation" of checking for acceptance of a regular expression for a violation

This involved modifying the DFS algorithm to

- Calculate states of the property automaton
- Check to see whether an accept state is reached

We will apply the same basic strategy for LTL

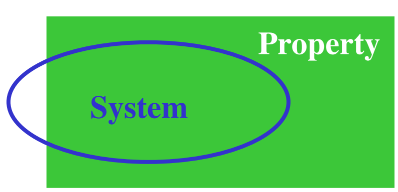
LTL Model Checking

From the semantics

- An LTL formula defines a set of (accepting) traces

We can

- Check for trace containment



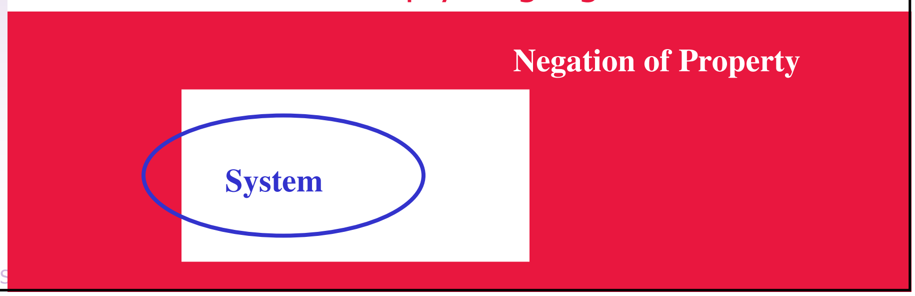
LTL Model Checking

From the semantics

- An LTL formula defines a set of (accepting) traces

We can

- Check for non-empty language intersection



Emptiness Check

LTL is closed under complement

$$\mathcal{L}(\phi) = \overline{\mathcal{L}(\neg\phi)}$$

where the language of a formula defines a set of *infinite* traces

A Buchi automaton accepts a set of infinite traces

Buchi Automata

A Buchi automaton is a quadruple (S, I, δ, F)

- S is a set of states
- $I \subseteq S$ is a set of initial states
- $\delta : S \rightarrow \mathcal{P}(S)$ is a transition relation
- F is a set of accepting states

Automaton states are labeled with atomic propositions of the formula

$\lambda : S \rightarrow \mathcal{P}(A)$

Example : Buchi Automaton

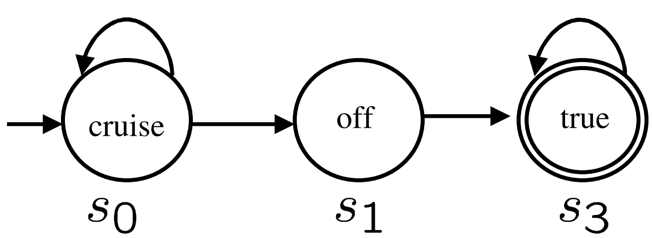
$S = \{s_0, s_1, s_2\}$

$I = \{s_0\}$

$\delta = \{(s_0, \{s_0, s_1\}), (s_1, \{s_2\}), (s_2, \{s_2\})\}$

$F = \{s_2\}$

$\lambda = \{(s_0, \{\text{cruise}\}, (s_1, \{\text{off}\}), (s_2, \{\})\}$



Buchi Automata Semantics

An infinite trace

$$\sigma = s_0 s_1 \ldots$$

is accepted by a Buchi automaton iff

$$s_0 \in I$$

$$\forall_{i \geq 0} : s_{i+1} \in \delta(s_i)$$

$$\forall_{i \geq 0} \exists_{j \geq i} : s_j \in F$$

Buchi Trace Containment

Assume each system state (S) is labeled (Λ) with the complete set of literals (A)

- either a literal or its negation is present

A Buchi automaton accepts a system trace

$\Sigma = S_0 S_1 \ldots$ iff

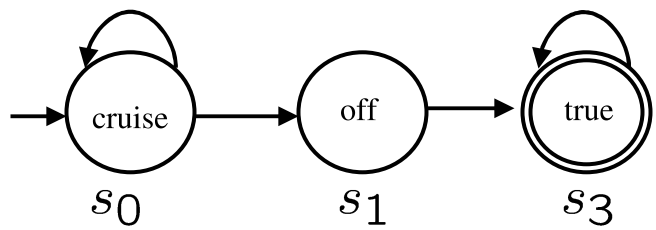
$\exists_{s_0 \in I} : \Lambda(S_0)$ satisfies $\lambda(s_0)$

$\forall_{i \geq 0} : \exists_{s_{i+1} \in \delta(s_i)} : \Lambda(S_{i+1})$ satisfies $\lambda(s_{i+1})$

$\forall_{i \geq 0} : \exists_{j \geq i} : s_j \in F$

Example : Buchi Automaton

$\sigma =$ cruise cruise off off accel accel cruise ...
$\sigma' =$ cruise cruise accel cruise off accell ...



LTL and Buchi Automata

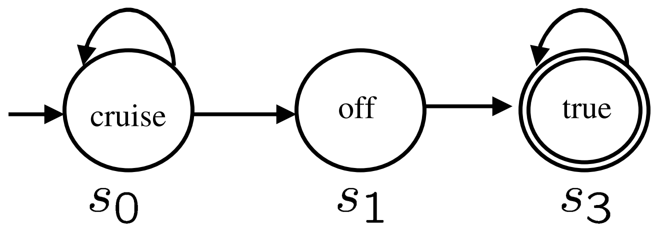
- Every LTL formula has a Buchi automaton that accepts its language (not vice versa)

  $\mathcal{L}(LTL) \subseteq \mathcal{L}(Buchi)$

  $\mathcal{L}(Buchi) \cap \mathcal{L}(LTL) \neq \emptyset$
- Buchi automata cannot be determinized
  - i.e., there is no canonical deterministic automaton that accepts the same language
- Buchi automata are closed under the standard set operations

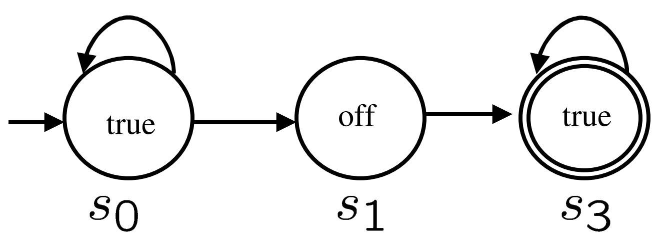
Example : Buchi Automaton

What LTL property does this correspond to?



Example : Buchi Automaton

What LTL property does this correspond to?



LTL Model Checking

- Apply same strategy as before
  - Generate Buchi automaton for the negation of the LTL property
  - Compose the automaton with the system
  - Check for emptiness
- Composition alternates transitions between the system and property
- Violation are indicated by accepting traces
  - Cycles containing an accept state

Nested DFS Algorithm

```
1 seen := {(s0, p0) | ∀p0∈I}
2 ∀p0∈I : DFS((s0, p0))

DFS(s, p)
3 workSet(s) := enabled(s)
4 while workSet(s) is not empty
5     let α ∈ workSet(s)
6     workSet(s) := workSet(s) \ {α}
7     s' := α(s)
7.1   if ¬∃p'∈δ(p) : Λ(s') satisfies λ(p') then
7.2       continue
8     if s' ∉ seen then
9         seen := seen ∪ {s'}
10        DFS((s', p'))
10.1      if p' ∈ F then
10.2         seen' = ∅
10.3         NDFS((s', p'), (s', p'))
end DFS
```

Multiple start states (search them all)

If you can't continue the property trace then give up (cannot lead to accept)

Only initiate a cycle check for accept states (since they are required in an acceptance cycle)

Nested DFS Algorithm

$NDFS((s,p)$, seed)
11 $workSet'(s)$:= *enabled*(s)
12 while $workSet'(s)$ is not empty
13 let $\alpha \in workSetN(s)$
14 $workSet'(s) := workSet'(s) \setminus \{\alpha\}$
15 $s' := \alpha(s)$
15.1 if $\neg\exists_{p' \in \delta(p)} : \Lambda(s')$ satisfies $\lambda(p')$ then
15.2 continue
16 if $(s', p') =$ seed then
17 Acceptance cycle detected
18 if $(s', p') \notin$ *seen'* then
19 *seen'* := *seen'* $\cup \{(s', p')\}$
20 $NDFS((s', p')$,seed)
end DFS

If you can't continue the property trace then give up (cannot lead to accept)

Any cycle is an acceptance cycle (since it started with an accept state)

For You To Do

- Take the dining philosophers example, and the property
  - [](P1.eating && P2.eating)
- Build a Buchi automaton for that property (using your intuition about automata)
- Apply the LTL NDFS algorithm
  - You may need to make the program counter explicit to do this since these automata are fundamentally state oriented
- Do you find an error?
- Can you think of a way to find errors faster in the NDFS() routine?

Fairness

- Progress states that the system should eventually do something
  - Often times in real systems threads rely on a schedule to give them a chance to run
  - Abstracting scheduling to non-deterministic choice introduces severe approximation
- There are many forms of fairness
  - The intuition is that we restrict the systems behaviors to only those on which each process gets a chance to execute

Fairness in LTL

- LTL is expressive enough to state fairness properties directly
  - []<> (Phil1.eating || Phil2.eating)
  - ([]<>Phil1.eating) && ([]<>Phil2.eating)
- Fairness formula can be used to *filter* the behaviors that are checked as follows
  - Fairness -> Property
  - If not Fairness then whole thing is true
  - Property checked only when Fairness holds