Software Model Checking: Theory and Practice

Lecture: Specification Checking - LTL Model Checking

Copyright 2004, Matt Dwyer, John Hatcliff, and Robby. The syllabus and all lectures for this course are copyrighted materials and may not be used in other course settings outside of Kansas State University and the University of Nebraska in their current form or modified form without the express written permission of one of the copyright holders. During this course, students are prohibited from selling notes to or being paid for taking notes by any person or commercial firm without the express written permission of one of the copyright holders.
Objectives

- To understand Büchi automata and their relationship to LTL
- To understand how Büchi 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

![Diagram of System and Property]
LTL Model Checking

From the semantics

- The negation of an LTL formula defines a set of (violating) traces

We can

- Check for non-empty language intersection
Emptiness Check

LTL is closed under complement

\[ L(\phi) = L(\neg \phi) \]

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

A Büchi automaton accepts a set of infinite traces
Büchi Automata

A Büchi automaton is a quadruple \( (R, I, \delta, F) \)

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

Unlike FSAs, Büchi automata are always non-deterministic

- set of initial states
- multiple transitions from a state
Büchi Automata

Automaton states are labeled with atomic propositions of the formula
\[ \lambda : R \rightarrow \mathcal{P}(A) \]
- where \( A \) are the set of observables for the program
- \( \lambda(r) \) is the set of observables for a property state

Note that the meaning of the automata is defined via this mapping
- plays the role of alphabet in FSA
Example: Büchi Automaton

\[ S = \{r_0, r_1, r_2\} \]
\[ I = \{r_0\} \]
\[ \delta = \{(r_0, \{r_0, r_1\}), (r_1, \{r_2\}), (r_2, \{r_2\})\} \]
\[ F = \{r_2\} \]
\[ \lambda = \{(r_0, \text{cruise}), (r_1, \{\text{off}\}), (r_2, \{\}\})\} \]
Büchi Automata Semantics

An infinite trace
\[ \sigma = r_0, r_1, \ldots \]
is accepted by a Büchi automaton iff
\[ r_0 \in I \]
starting in an initial state
\[ \forall i \geq 0 : r_{i+1} \in \delta(r_i) \]
trace corresponds to transition relation
\[ \forall i \geq 0 \ \exists j \geq i : r_j \in F \]
can reach a final state from end of all prefixes
Büchi Trace Containment

Assume each system state \( S \) is labeled \( \Lambda \) with set of observables \( A \)

A Büchi automaton accepts a system trace \( s_0, s_1, \ldots \)

\[
\exists r_0 \in I : \lambda(r_0) \in \Lambda(s_0)
\]

\[
\forall i \geq 0 \ \exists r_{i+1} \in \delta(r_i) : \lambda(r_{i+1}) \in \Lambda(s_{i+1})
\]

\[
\forall i \geq 0 \ \exists j \geq i : r_j \in F
\]
Example: Büchi Automaton

cruise cruise off off accel accel cruise …
cruise cruise accel cruise off accel …

\[
\begin{array}{c}
cruise \\
r_0 \\
\end{array}
\quad \quad
\begin{array}{c}
off \\
r_1 \\
\end{array}
\quad \quad
\begin{array}{c}
true \\
r_2 \\
\end{array}
\]
LTL and Büchi Automata

- Every LTL formula has a Büchi automaton that accepts its language (not vice versa)
  \[ L(\phi) \subseteq L(\text{BA}) \]
  \[ L(\phi) \cap L(\text{BA}) \neq \emptyset \]

- Büchi automata cannot be determinized
  - i.e., there is no canonical deterministic automaton that accepts the same language

- Büchi automata are closed under the standard set operations
Example: Büchi Automaton

What LTL property does this correspond to?

\[ \text{cruise} \mathcal{U} \text{off} \]
Example: Büchi Automaton

What LTL property does this correspond to?

◊ off

◊ off
LTL Model Checking

- Apply same strategy as before
  - Generate Büchi automaton for the negation of the LTL property
  - Explore state space of the product of the automaton and the system
  - Check for emptiness

- Violation are indicated by accepting traces
  - Look for cycles containing an accept state
  - Use nested depth-first search
LTL Model Checking

For each initial property state
initialize DFS data structures
perform search of initial product state
LTL Model Checking

DFS((s,r))

workSet := enabled(s)

for each \( \alpha \in \text{workSet} \) do

\( s' := \alpha(s) \)

for each \( r' \in \delta(r) \) do

if \( \lambda(r') \notin \Lambda(s') \) then

if \( (s',r') \notin \text{seen} \) then

\( \text{seen} := \text{seen} \cup \{(s',r')\} \)

push(stack, (s',r'))

DFS((s',r'))

if \( r' \in A \) then

\( \text{seen}' := \{(s',r')\} \)

stack' := [(s',r')]

NDFS((s',r'),(s',r'))

pop(stack')

end DFS

For each transition check if state labels match

Only launch a cycle search from property accept states
For each transition
check if state labels match

Same logic as for progress checking
Fairness

- Liveness 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
  - $[\neg\neg(\text{Phil1.eating} \lor \text{Phil2.eating})]$
  - $([\neg\neg\neg\neg(\text{Phil1.eating}) \land \neg\neg\neg\neg(\text{Phil2.eating})])$
- 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