

COMP6463: Temporal Logic and Model Checking

6. LTL Model Checking

Michael Norrish

Canberra Research Lab., NICTA

Semester 2, 2010



Outline

1 Introduction

2 LTL

- Syntax and Semantics

- Büchi Automata

- Model-Checking LTL with Büchi Automata

Why Other Logics?

CTL is great because it's **easy to check**.

CTL is not so great because it **can't express everything we might want**.

What are some other candidate logics?

- How hard are they to check?
- What cool techniques arise?

LTL: Linear Temporal Logic

In **CTL**, had quantifiers that were fixed combinations of

- path quantifiers: **A** and **E**
- quantifiers on states in paths: **X**, **F**, **G**, **U**

In **LTL**, drop path quantifiers and allow arbitrary combinations of **X**, **F**, **U** and **G**.

(If you put path quantifiers back in, but allow arbitrary combinations, you get **CTL***.)

LTL Syntax

Well-formed LTL formulas:

$$\begin{aligned}\phi &::= p, q, r \dots \\ &| \neg \phi \\ &| \phi_1 \vee \phi_2 \\ &| X \phi \\ &| \phi_1 U \phi_2\end{aligned}$$

Define \wedge , \Rightarrow , \perp , \top in usual propositional way.

Also

$$\begin{aligned}F \phi &\equiv \top U \phi \\ G \phi &\equiv \neg F \neg \phi\end{aligned}$$

Sometimes $F = \diamond$, $G = \square$, $X = \bigcirc$.

Path Suffixes

The semantics of LTL will be with respect to models and **paths** through that model

- **not** individual worlds as in CTL

Recall: a **path** π is a function $\mathbb{N} \rightarrow \text{state}$

Let π^i be the suffix of π starting from $\pi(i)$.

I.e.,

$$\pi^i(x) = \pi(x + i)$$

Evaluating LTL

An LTL formula is typically held to be true of a frame if it's true for **every path** in that frame.

People will sometimes put an **A** in front to make this clear.

E.g., $A(GF \phi)$

The **A** can never appear anywhere else.

LTL Semantics

\mathfrak{M} is a triple (W, R, V) , with

- W a set of worlds;
- R a binary relation on those worlds;
- V a function from worlds to sets of propositional letters

π is an R -path through W .

$\mathfrak{M}, \pi \models p$ if $p \in V(\pi(0))$

$\mathfrak{M}, \pi \models \neg\phi$ if $\mathfrak{M}, \pi \not\models \phi$

$\mathfrak{M}, \pi \models \phi_1 \vee \phi_2$ if $\mathfrak{M}, \pi \models \phi_1$ or $\mathfrak{M}, \pi \models \phi_2$

$\mathfrak{M}, \pi \models X\phi$ if $\mathfrak{M}, \pi^1 \models \phi$

$\mathfrak{M}, \pi \models \phi_1 \cup \phi_2$ if there exists a i such that $\mathfrak{M}, \pi^i \models \phi_2$
and for all $0 \leq j < i$, $\mathfrak{M}, \pi^j \models \phi_1$

Expressivity

LTL and CTL can each express things that the other cannot.

LTL has

$$A(FG p)$$

which means “there is a point on every path from which p holds forever”.

CTL has

$$AG(EF p)$$

which means “on every path, at every point, there is a path to a point where p ” is true.

Model Checking LTL

Surprisingly (?), LTL is (probably) harder to check.

LTL is **PSPACE**-complete.

(So too is CTL*, which is also curious.)

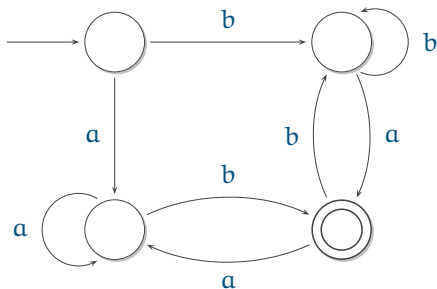
One method for checking LTL involves translation to Büchi Automata.

Büchi Automata

A **Büchi automaton** is a finite state automaton extended to accept/reject **infinite** strings.

To be accepted, an infinite string must cause the automaton to enter an accepting state infinitely often.

This automaton accepts those strings where **a** and **b** both appear infinitely often.

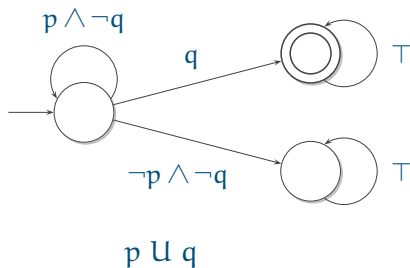
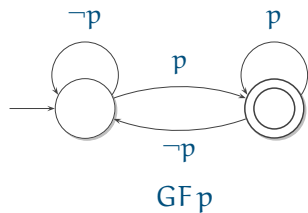
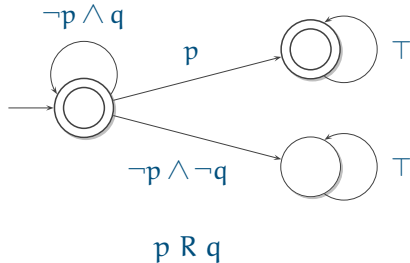
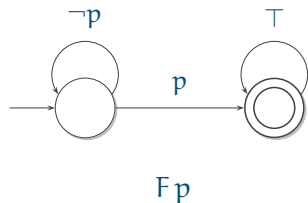


Model Checking with Büchi Automata

Four key properties:

- LTL properties can be turned into Büchi automata
 - Labels on arrows are propositional formulas
 - In fact, Büchi automata can capture properties not expressible in LTL
- Kripke frames can be turned into Büchi automata
 - All states accept (if fairness constraints do not intrude)
 - Labels on arrows are propositional atoms true in destination
- The intersection of two Büchi automata is a Büchi automaton
- It is easy (linear time) to check if a Büchi automaton accepts any string at all
 - Identify the reachable strongly-connected components; if any includes an accepting state, return “yes”

LTL Formulas as Büchi Automata



Frames as Büchi Automata

- Add an extra `init` state with outgoing edges to all initial states
- Label edges with the conjunction of the propositional literals true at the destination state
- Make every state accepting

Intersections of Büchi Automata

The intersection of

- $(Q_1, s_1, \rightarrow_1, F)$ (derived from an LTL formula), and
- $(Q_2, s_2, \rightarrow_2, U)$ (derived from a frame)

is

$$(Q_1 \times Q_2, (s_1, s_2), \rightarrow, \{(x, y) | x \in F\})$$

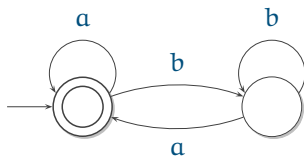
where \rightarrow puts an ℓ -edge between (x_0, y_0) and (x, y) if

- $x_0 \xrightarrow{\ell}_1 x$, and
- $y_0 \xrightarrow{\ell}_2 y$

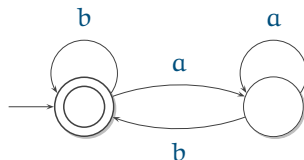
Fairness Complicates Matters

The definition of intersection becomes more complicated when not all of the Kripke frame's states are accepting, because of fairness.

For example, intersect these two naïvely



“infinite number of **a**”



“infinite number of **b**”

and the result is a Büchi automaton that accepts nothing.

Model-Checking LTL—Take 1

Problem: “is formula ϕ true of model M ?”

- 1 Convert M to automaton A_m
- 2 Convert ϕ to automaton A_ϕ

Question is then: “is $\mathcal{L}(A_m) \subseteq \mathcal{L}(A_\phi)$?”

Which is the same as: “is $\mathcal{L}(A_m) \cap \overline{\mathcal{L}(A_\phi)} = \emptyset$?”

Model-Checking LTL—Take 1

Problem: “is formula ϕ true of model M ?”

- 1 Convert M to automaton A_m
- 2 Convert ϕ to automaton A_ϕ

Question is then: “is $\mathcal{L}(A_m) \subseteq \mathcal{L}(A_\phi)$?”

Which is the same as: “is $\mathcal{L}(A_m) \cap \overline{\mathcal{L}(A_\phi)} = \emptyset$?”

We can calculate intersection, and can check for emptiness.

But calculating the complement of a Büchi automaton, though possible, is ugly...

Model Checking LTL

Problem: “is formula ϕ true of model M ?”

- 1 Convert M to automaton A_M
- 2 Convert $\neg\phi$ to automaton $A_{\neg\phi}$
- 3 Check $\mathcal{L}(A_M) \cap \mathcal{L}(A_{\neg\phi}) = \emptyset$
 - if yes, then property is true
 - if no, then property is false

LTL Counter-Examples

If formula ϕ was not true of model \mathcal{M} , then

$$\mathcal{L}(A_m) \cap \mathcal{L}(A_{\neg\phi}) \neq \emptyset$$

In other words, the intersection automaton accepts an infinite string.

In other words, there is a reachable strongly-connected component in the intersection automaton.

The path leading to this component, and the looping path within it, together constitute a **counter-example** to the desired property.

On-the-fly Model Checking

CTL model-checking with BDDs

- requires a BDD for the whole transition relation to be constructed
- (This can make model-checking very expensive.)

We earlier saw how the (incomplete) Java Pathfinder checker explored the state-space of a program incrementally.

- This made “checking” the (presumably infinite) state-space possible.

On-the-fly Model Checking for LTL

The Büchi automaton for a model need not be constructed all at once.

Recall three steps:

- 1 construct $A_{\neg\phi}$
- 2 construct A_M
- 3 check emptiness of $A_{\neg\phi} \cap A_M$

Checking emptiness is effectively a search for a loop (that includes an accepting state) which can be done with depth-first traversals.

Only construct $A_{\neg\phi} \cap A_M$ as the DFS proceeds. . .

Lazy Construction of Intersection Automaton

Imagine search is on node $\langle q, r \rangle$, with $q \in A_{\neg\phi}$, and $r \in A_M$.

Construct neighbours (“successor nodes”) of r (in A_M), and only explore those that share the appropriate labels with $A_{\neg\phi}$.

Algorithm terminates:

- when a loop is found: which may happen well before all of A_M is explored
- **or** when all of state space is explored.

LTL Implemented in SPIN Tool

See <http://www.spinroot.com>.

- Provides PROMELA modelling language
- Models can be interactively simulated by user (to explore design manually).
- LTL properties can be checked of the system
- Clever hashing of states gives very good coverage when exhaustive verification is not possible

LTL Summary

LTL is a “linear time” temporal logic.

It can be checked efficiently by conversion (of formulas and models) to Büchi automata.

It can do “on the fly” checking, avoiding need to expand all of model into automata states.

Implemented (most famously) in SPIN tool.

Model Checking Summary

Model Checking is a Huge Subject

- Great example of **theory** applied to **practice**
- Opportunities both to do some elegant maths. . .
- . . . and to implement/engineer that maths for the benefit of system designers everywhere.