Symbolic Model Checking

Computer Science and Artificial Intelligence Laboratory
MIT
Armando Solar-Lezama

Dec 06, 2011

Review of Temporal Logic

Engine starts and stops with button push

- If engine is off, it stays off until I push
  - If I never push it stays on forever
- If engine is on, it stays on until I push
  - If I never push it stays off forever

`on, off, push, id`

$$G\ off \Rightarrow off\ U\ push$$

$$G\ (off \Rightarrow (off\ U\ push \lor G\ off))$$

$$G\ (on \Rightarrow (on\ U\ push \lor G\ on))$$

The problem with Explicit State MC

There are too many states

- way, way too many states

explicit state MC can only scale to about 10^20 states

- that’s not enough for many systems

Symbolic Model Checking

Don’t store the state graph

- keep instead a symbolic representation of the state transition system

This was a big idea

- Ken McMillan

Key Idea 1: Sets and boolean algebra

There is a close connection between set theory and logic

Set Theory	First Order Logic
set $S=\{x_1, \ldots, x_n\}$	predicate P_S s.t. $P_S(x_i) := \text{true}$
set union $S \cup E$	disjunction (P_S or P_E)
set intersection $S \cap E$	conjunction (P_S and P_E)
empty set Ø	$P_{\emptyset} = \text{false}$
subset $S \subseteq E$	implication $P_S \rightarrow P_E$

Key Idea 2: Predicates as boolean circuits

Predicate P_s is defined on a finite universe of symbols X

We can represent each element of X with a bit-vector

- we need only log |X| bits per element

With this representation, P_s can be defined as a circuit

Ex.

- Let X be the set of integers between 0 and $2^{32}-1$
- $P_{even}(x) = (\text{not } x_{lsb})$

Key Idea 3: Automata and Sets

Automata are defined in terms of sets

- Kripke Structure = (S, S_0, R, L)

- S : Universe of possible states
  - One bit-vector per element of S.
- S0 defined by a predicate P_{S0}

- R: is a relation, i.e. a set of pairs (s_i, s_{i+1})
  - $P_R\ (s_i, s_{i+1})$

Key Idea 4: Decision Procedures

We have really good procedures for boolean logic

- BDDs were state of the art in 1990
- SAT is more common today
  - BDDs still good for niche applications
- SMT is rapidly becoming the norm
  - Satisfiability Modulo Theories
  - combines SAT with decision procedures for:
    - integers, arrays, uninterpreted functions, …

BDDs

Compact representation of a binary tree

- Remove redundancies
- Share nodes

Easy to run certain kinds of queries

- Emptyness, boolean operations

They can blow up!

Checking Safety Properties

Suppose we want to check the property G p

Strategy:

- compute the set of reachable states S_{reach}
- check if an element of S_{reach} satisfies (not p)

How do we compute S_{reach}?

Checking Safety Properties

Let S_i be the set of states reachable after i steps

- What's the relationship between S_i and S_{i-1}?

We can define P_{Si+1} as

- $P_{Si+1}(v) = P_{Si}(v)$ or $\exists x \{ P_{si}(x)$ and $R(x, v)\}$
- This is a recursive definition
- We can find $P_{S\infty}$ by iteratively computing P_{si} until we find a fixed point
  - $P_{S1}(x) = P_{S0}(x)$ or $(P_{S0}(x_0)$ and $R(x_0, x))$
  - $P_{S2}(x) = P_{S1}(x)$ or $(P_{S0}(x_0)$ and $R(x_0, x_1)$ and $R(x_1, x))$
  - $P_{S3}(x) = P_{S2}(x)$ or $(P_{S0}(x_0)$ and $R(x_0, x_1)$ and $R(x_1, x_2)$ and $R(x_2, x))$

Checking Safety Properties

Two big questions

- How do we know if we have reached a state where (not p)?
  - that's easy
  - we can assume a predicate $P_p(x)$ that is true for any state where p holds
  - x is a reachable bad state if (not $P_p(x)$) and $P_{Si}(x)$
- How do we know when we have explored all reachable states?
  - when $P_{si} = P_{si+1}$
  - i.e. not P_{si} (x) and (P_{si+1}(x)) becomes unsatisfiable

The challenge

- Can we generalize this to work for arbitrary formulas?

Checking General CTL Formulas

Why CTL

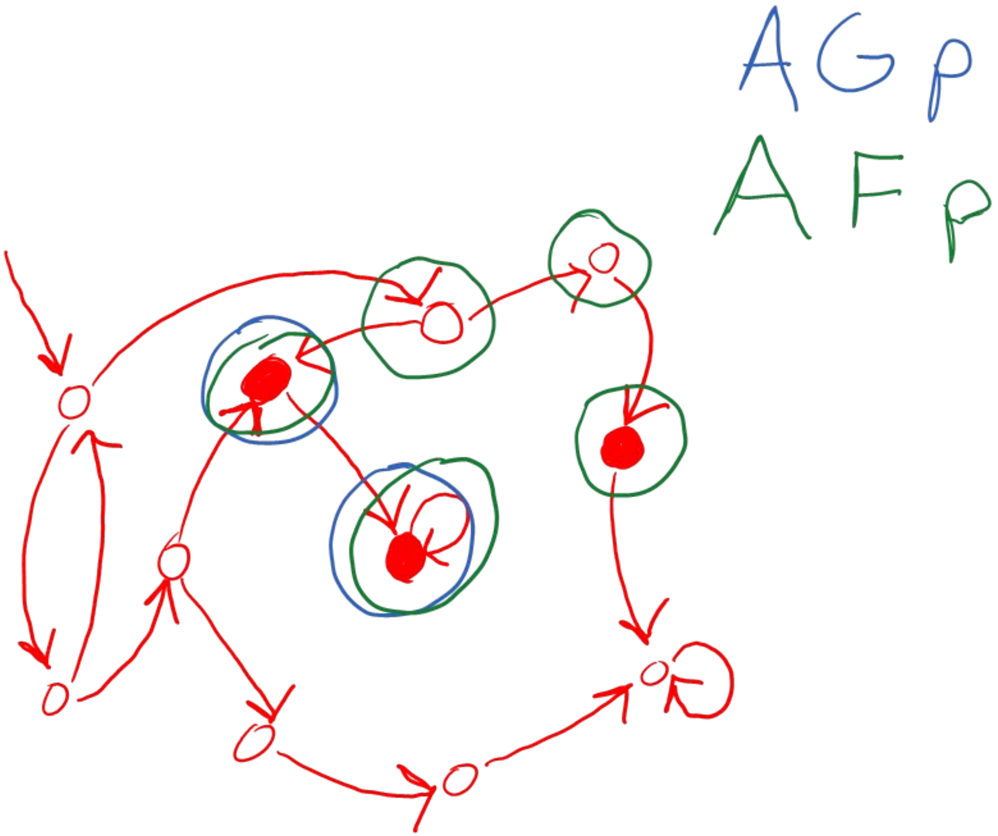
- it's "easy"

We'll consider only the following formulas:

- p ::= E X p | E G p | E (p U q) | p binop q

Basic Intuitions

We can map CTL formulas to the states where the formula holds



Basic Intuitions

We can map CTL formulas to the set of states where the formula holds

Sets of states == Boolean formula

- We can recursively map CTL formulas to boolean formulas

Model Checking CTL properties

We will do it with a recursive CHECK procedure

- Input: A CTL property P
- Output: A boolean formula representing the states that satisfy P

Cases

- P is a boolean formula: Check(P) = P
- P = EX p, then Check(P) = CheckEX(Check(p))
- P = E p U q, then Check(P) = CheckEU(Check(p), Check(q))
- P = E G p, then Check(P) = CheckEG(Check(p))

CheckEX

CheckEX(p) returns a set of states such that p is true in their next states

- So if $CheckEX(p) \equiv Q$ then $Q(x) \equiv \exists x' \; s.t. R(x, x') \wedge p(x')$

CheckEU

CheckEU(p, q) returns a set of states such that

- Either q is true in that state or
- p is true in that state and you can get from it to a state in which E(p U q) is true
- $Z_k(v) = (q(v) \lor [p(v) \land \exists v' R(v, v') \land Z_{k-1}(v')]$
- $Z_0(v) = false$
- CheckEU(p,q) $\equiv Z_\infty$

CheckEG

What about CheckEG(p)

- p is true in the current state and you can get from this state to another state where EG(p) is true
- $Z_k(v) = p(v) \wedge \exists v' R(v, v') \wedge Z_{k-1}(v')$
- $Z_0(v) = true$
- CheckEG(p) $\equiv Z_\infty$

How do we know these formulas are well defined?

Fixpoints

Let Σ be a set with $\Sigma' \subseteq \Sigma$

Let τ: $P(\Sigma) \rightarrow P(\Sigma)$

Some properties:

- Σ' is a fixpoint if $\tau(\Sigma') = \Sigma'$
- τ is monotonic iff $P \subseteq Q \Rightarrow \tau(P) \subseteq \tau(Q)$
- τ is U-continuous iff $P_1 \subseteq P_2 \subseteq P_3 \subseteq \ldots \Rightarrow \tau(\cup P_i) = \cup\, \tau(P_i)$
- τ is ∩-continuous iff $P_1 \subseteq P_2 \subseteq P_3 \subseteq \ldots \Rightarrow \tau(\cap P_i) = \cap\, \tau(P_i)$

Main theorem

- A monotonic τ always has a least fixed point:

$$\mu Z.\ \tau(Z) = \cap\{ Z \mid \tau(Z) \subseteq Z\}$$

$$= \cap\ \tau^i(\Sigma) \text{ when } \tau \text{ is } \cap\text{-continuous}$$

- A monotonic τ always has a greatest fixed point:

$$\nu Z.\ \tau(Z)=\cup\{ Z \mid \tau(Z) \supseteq Z\}$$

$$= \cup\ \tau^i(\emptyset) \text{ when } \tau \text{ is U-continuous}$$

Fixpoints

If $\boldsymbol{\Sigma}$ is finite, and τ is monotonic,

then it is τ is ∩-continuous and U-continuous

CTL in terms of fixpoints

Given a CTL formula, we want to characterize the set of states that satisfy the formula

A G p = ν Z. τ(Z) where τ(Z) = p and A X Z

6.820 Fundamentals of Program Analysis
Fall 2015