6894: Lightweight Formal Methods

Lecture 13: Model checking, Part 1.

April 4, 2005

Daniel Jackson.

Last week:

temporal logics, CTL and LTL.

This week:

model checkers

technology — how they work

notation — especially describing the machine.

Today

symbolic model checking.

The model checking problem:

Given machine M, temporal logic formula F,
find set of states

$$\{s \in S \mid M, s \models F\}$$

(where M is a Kripke structure $\langle S, R, L \rangle$
whose propositions L appear in F).

CTL Checking.

Basic idea is to label states with subformulas of F.
Let label(s) be formulas in F true at s.
Initially, label(s) $= L(s)$
In ith stage, subformulas with $i-1$ nested CTL operators are added;
At the end $M, s \models F$ iff $F \in$ labels(s).

Any CTL formula can be expressed with

$$\neg, \vee, EX, EU, EG$$

so we handle these forms inductively.

For $\neg f$: Label with $\neg f$ all states not labelled with f

$f \vee g$ $f \vee g$ labelled with f or g.

$E[f \, U \, g]$

1. find states labelled with g.
2. follow $\sim * R$ and find all states reachable by path labelled by f in every state.

EGf is harder.

First, break graph into non-trivial strongly connected components.

SCC of graph g : maximal subgraph C such that every node in C is reachable from every other node in C along a path contained within C.

non-trivial: has >1 node, or a self loop.

Let M' be obtained from M by deleting all states at which f does not hold and restricting R and L accordingly. Now note:

$M, s \models EGf$ iff

① $s \in S'$

② exists path in M' leading from s to some node t in a non-trivial SCC of (S', R').

So algorithm is: 1. drop all states not labelled f.
2. break into non-trivial SCCs.
3. Label all states in SCC's with EGf
4. for each state s in ~~the reduced machine~~: T
   for all predecessors of s
   if not labelled EGf, label and ~~add~~ add to T.

Here are the algorithms: (from Clarke, Grumberg, Peled).

```
CheckEU (f1, f2)
    T := {s | f2 ∈ label(s)}
    for all s ∈ T do
        label(s) := label(s) ∪ {E[f1 U f2]}
    while T ≠ ∅ do
        pick s ∈ T
        T := T − {s}
        for all t such that R(t,s) do
            if E[f1 U f2] ∉ label(t) and f1 ∈ label(t)
                label(t) := label(t) ∪ {E[f1 U f2]}
                T := T ∪ {t}
```

```
CheckEG (f)
    S' := {s | f ∈ label(s)}
    SCC := {C | C is a non-trivial scc of S'}
    T := ⋃_{c∈SCC} {s | s ∈ C}
    for all s ∈ T do
        label(s) := label(s) ∪ {EGf}
    while T ≠ 0 do
        pick s ∈ T
        T := T − {s}
        for all t such that t ∈ S' and R(t,s) do
            if EGf ∉ label(t)
                label(t) := label(t) ∪ {EGf}
                T := T ∪ {t}
```

Complexity:

Tarjan's SCC algorithm: $O(|S| + |R|)$
CheckEG and checkEU $O(|S| + |R|)$

Need one pass for each subformula of F,
so overall complexity is

$$O(|f| \cdot (|S| + |R|))$$

i.e. it's LINEAR in size of formula and size of machine.
problem is that $|S|$ can be exponential in the size of the description of the machine.

Example: Microwave oven from [CAP], p38-39.

Symbolic Model checking.

Idea: instead of examining states one at a time, manipulate large sets of states all together. Rather than representing the states explicitly (eg. in a data structure that has an element for each state), represent them symbolically as a formula.

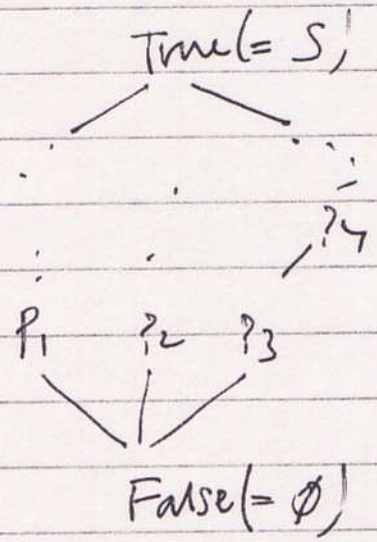
To make this work, we need to be able to compare two formulas (to check that a set of states being visited is no longer growing). This requires a canonical representation.

To understand sym. model checking then, we'll need to
(i) recast the algorithms in terms of sets of states;
(ii) see how to represent sets of states symbolically, in a canonical form.

Symbolic Model checking

Given a Kripke structure $M = \langle S, R, L \rangle$, any subset of the set of states S can be viewed as a predicate on states.

The predicates form a lattice



A predicate transformer is a function from predicates to predicates:

$$\tau : \mathbb{P}S \to \mathbb{P}S$$

Least fixpoint, written $\mu Z.\ \tau(Z)$ is smallest predicate P such that $P = \tau(P)$

Greatest fixpoint, written $\nu Z.\ \tau(Z)$, is largest predicate P such that $P = \tau(P)$.

Some theory due to Tarski:

τ is monotonic iff $P \subseteq Q \Rightarrow \tau(P) \subseteq \tau(Q)$

$\cup$-continuous $P_1 \subseteq P_2 \subseteq \ldots \Rightarrow \tau(\cup_i P_i) = \cup_i \tau(P_i)$

$\cap$-continuous $P_1 \supseteq P_2 \supseteq \ldots \Rightarrow \tau(\cap_i P_i) = \cap_i \tau(P_i)$

if τ is monotonic and $\cup$-continuous:

$$\mu Z.\tau(Z) = \cup_i \tau^i(\text{False})$$

if τ is monotonic and $\cap$-continuous:

$$\nu Z.\tau(Z) = \cap_i \tau^i(\text{True})$$

For finite structure, τ is monotonic $\Rightarrow$ $\cup$- and $\cap$-continuous.
Also, if τ is monotonic,

$$\tau^i(\text{False}) \subseteq \tau^{i+1}(\text{False})$$
$$\tau^i(\text{True}) \supseteq \tau^{i+1}(\text{True})$$

So can compute like this:

```
fun lfp (tau : pred → pred) : pred
    Q := False
    Q' := tau(Q)
    while Q ≠ Q' do
        Q := Q'
        Q' := tau(Q')
    return Q.
```

```
gfp (..) ..
    Q := True
    ..
```

only difference

Now the remarkable bit.

Each CTL operator can be expressed as a gfp or lfp.

$$AF\,f = \mu Z.\,(f \vee AX.Z)$$
$$EF\,f = \mu Z.\,(f \vee EX.Z)$$
$$AG\,f = \nu Z.\,(f \wedge AX.Z)$$
$$EG\,f = \nu Z.\,(f \wedge EX.Z)$$

$$A[f\,U\,g] = \mu Z.\,(g \vee (f \wedge AX.Z))$$
$$E[f\,U\,g] = \mu Z.\,(g \vee (f \wedge EX.Z))$$

Note:

lfp's for eventualities

gfp's for always

Symbolic Computation.

Model checking is implemented by an algorithm Check that takes a CTL formula and returns the set of states satisfying it, represented as a predicate.

```
Check EXf    = Check_EX (Check(f))
Check E[f U g] = CheckEU (Check(f), Check(g))
Check EGf    = CheckEG (Check(f))
```

↖ These helper procedures take predicates as arguments.

How do these helpers work?

$$\text{Check EX } f\ P = \exists s'.\ P(s') \wedge R(s, s')$$

[We'll see this later: it's called the "weakest precondition" of P]

CheckEU and CheckEG use the fixpoint algorithms.

eg. $E[f\ U\ g] = \mu Z.(g \vee (f \wedge EX\text{-}Z))$

So the only remaining problem is how to represent the predicates so that

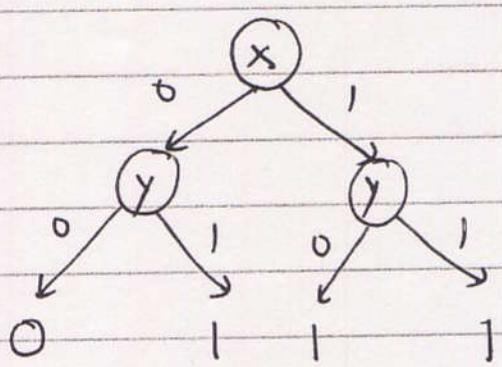
(i) we can apply predicate transformers and

(ii) we can do the $Q = Q'$ test.

Binary Decision Diagrams

Actually OBDD's
↖ ordered - very important!
idea due to Randy Bryant.

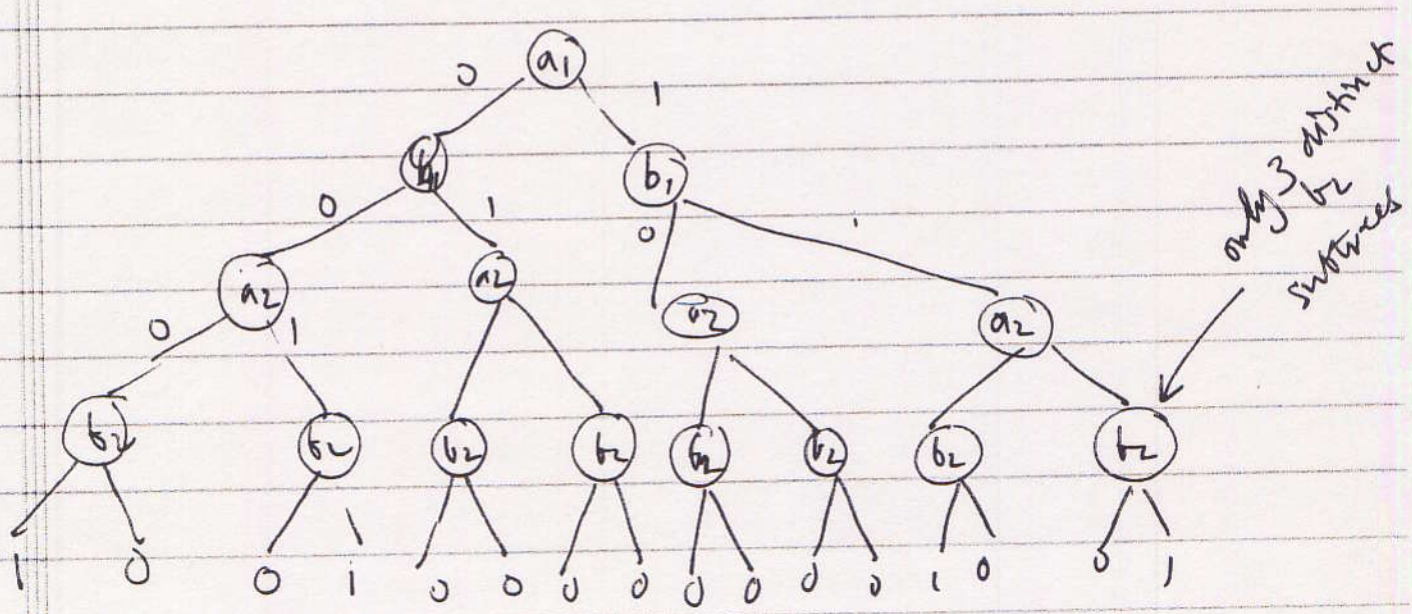
First, binary decision trees.
a form of truth table



x ∨ y.

Often lots of redundancy.
Consider 2-bit comparator

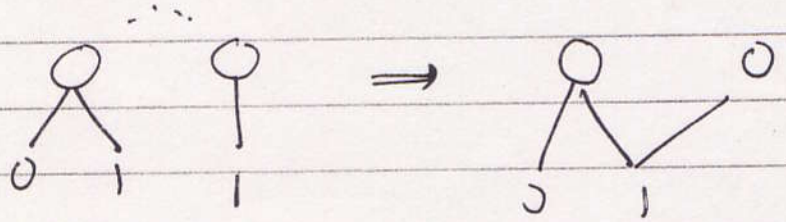
$$f(a_1, a_2, b_1, b_2) = a_1 \Leftrightarrow b_1 \wedge a_2 \Leftrightarrow b_2$$



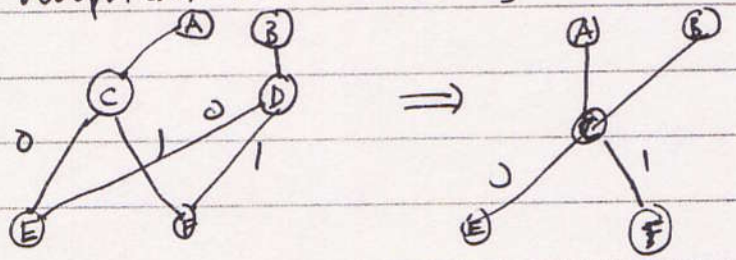
Binary decision diagram
↳ merge isomorphic subtrees.

Finally, OBDDs:
make the diagrams CANONICAL with 2 rules:
1. same variable ordering.
2. no isomorphic subtrees or redundant vertices.

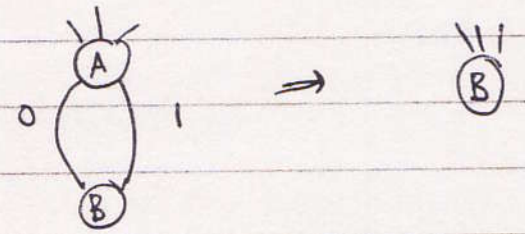
Removing redundancy:
duplicate terminals



duplicate non-terminals



redundant tests



Can implement all logical ops on OBDDs.

How do we represent Kripke structures as OBDD's?

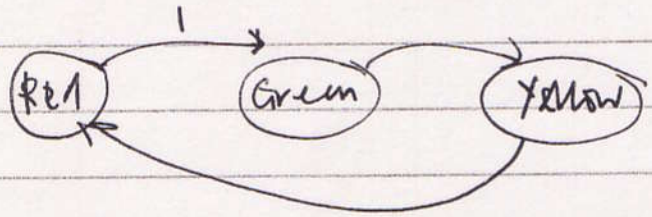
↳ variables are labels L

↳ set of states is just a propositional formula over L.

transition relation is a formula on

$\langle L_1, L_2, L_3, \ldots L_n, L_1', L_2', L_3', \ldots L_n' \rangle$

Eg.



$(R \wedge \bar{G} \wedge \bar{Y} \wedge G' \wedge \bar{R}' \wedge \bar{Y}')$ is transition 1.

transition relation is just disjunction of transitions.

$R\bar{G}\bar{Y}\,G'\bar{R}'\bar{Y}'$

$\vee \quad G\bar{R}\bar{Y}\,Y'\bar{G}'\bar{R}'$

$\vee \quad Y\bar{G}\bar{R}\,R'\bar{G}'\bar{Y}'$

What's the snag with SMC?

OBDDs can grow very large.

* Size very sensitive to var ordering.
* Finding the best ordering is infeasible;
  checking that one ordering is optimal is NP-complete.
* there are boolean functions that have exponential size OBDDs for any ordering of variables.

So in practice, the var ordering must often be manually tweaked.