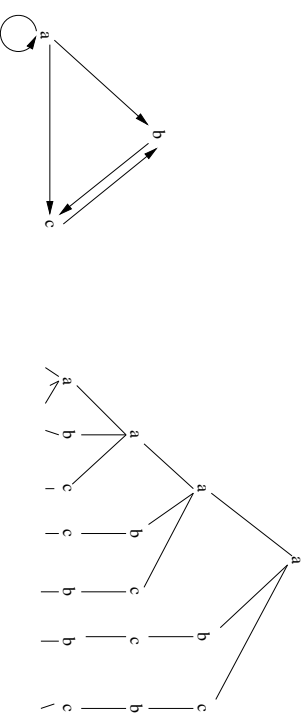


Logic Summer School, ANU Dec 2003
 Formal Methods 1: Algorithmic Verification
 Lecture 4: Branching Time Temporal Logic, BDD's
 Ron van der Meyden (UNSW/NICTA)

Suppose that $M = \langle S, S_0, \rightarrow, L \rangle$ is a transition system.
 A *path* in M is a sequence of states s_0, s_1, \dots such that $s_i \rightarrow s_{i+1}$ for all i .

Computation Trees



Computation Tree Logic — CTL*

Syntax: two types formulas:

state formulas:

1. p , where $p \in Prop$
2. $\neg\phi_1$ and $\phi_1 \wedge \phi_2$, where ϕ_1, ϕ_2 are state formulas
3. $\mathbf{E}\phi$, where ϕ is a path formula

path formulas:

1. ϕ , where ϕ is a state formula
2. $\neg\phi_1$ and $\phi_1 \wedge \phi_2$, where ϕ_1, ϕ_2 are path formulas
3. $\mathbf{X}\phi_1$ and $\phi_1 \mathbf{U} \phi_2$ where ϕ_1, ϕ_2 are path formulas

CTL* Semantics

Given a transition system M , we interpret path formulas on paths r in M and state formulas at states s in M :

Path formulas:

1. $M, r \models \phi$, where ϕ is a state formula, if $M, s \models \phi$
 2. $M, r \models \neg\phi$ if not $M, r \models \phi$
 3. $M, r \models \phi_1 \wedge \phi_2$ if $M, r \models \phi_1$ and $M, r \models \phi_2$
 4. $M, r \models \mathbf{X}\phi_1$ if $M, r^1 \models \phi_1$
 5. $M, r \models \phi_1 \mathbf{U} \phi_2$ if for some $k \geq 0$ we have $M, r^k \models \phi_2$ and for $0 \leq l < k$ we have $M, r^l \models \phi_1$
- where $r^k = r(k), r(k+1), r(k+2), \dots$

State formulas:

1. $M, s \models p$, where $p \in Prop$ if $p \in \pi(s)$
2. $M, s \models \neg\phi$ if not $M, s \models \phi$
3. $M, s \models \phi_1 \wedge \phi_2$ if $M, s \models \phi_1$ and $M, s \models \phi_2$
4. $M, s \models \mathbf{E}\phi$ if there exists a path r in M with $r(0) = s$ such that $M, r \models \phi$

Examples

Abbreviation: $\mathbf{A}\phi$ for $\neg\mathbf{E}\neg\phi$ (ϕ holds on all paths)

At all times, one of the two processes must be able to proceed:

$$\mathbf{AG}(\mathbf{EX}moved_1 \vee \mathbf{EX}moved_2)$$

It is always possible to get to the *restart* state

$$\mathbf{AG}(\mathbf{EF}restart)$$

If process 1 reaches the trying state, it will eventually reach the critical state

$$\mathbf{AG}(loc_1 = \text{try} \Rightarrow \mathbf{AF}(loc_1 = \text{crit}))$$

CTL

All the above formulas are in fact in a fragment CTL of CTL*:

CTL is the fragment of CTL* where we restrict the set of path formulas to be of the forms $\mathbf{X}\phi_1, \phi_1 U \phi_2, \neg(\phi_1 U \phi_2)$ where ϕ_1, ϕ_2 are state formulas.

(This includes $\mathbf{F}\phi_1 = \text{true U } \phi_1$ and $\mathbf{G}\phi_1 = \neg(\text{true U } \neg\phi_1)$.)

An alternate syntactic basis for CTL path formulas is: $\mathbf{X}\phi_1, \phi_1 U \phi_2, \mathbf{G}\phi_1$, since

$$\mathbf{E}\neg(\phi_1 U \phi_2) \equiv (\mathbf{EG}\neg\phi_2) \vee \mathbf{E}(\neg\phi_2) \mathbf{U} (\neg\phi_1 \wedge \neg\phi_2))$$

Model checking CTL state formulas

Theorem: For $\phi \in CTL$, determining $M, s \models \phi$ can be done in time $O(|M| \cdot |\phi|)$.

Let S' be the set of reachable states of E reachable from s via \rightarrow .
label states in S' by subformulas of ϕ , proceeding from leaves of the parse tree to the root

1. label s by p if $p \in \pi(s)$
2. label s by $\neg\phi_1$ if s not labelled by ϕ_1
3. label s by $\phi_1 \wedge \phi_2$ if s labelled by both ϕ_1, ϕ_2
4. label s by **EX** ϕ_1 if for some state t we have $s \rightarrow t$ and t labelled by ϕ_1
5. label s by **E**(ϕ_1 **U** ϕ_2) if there exists a sequence $s = s_0, s_1, \dots, s_k$ such that s_k labelled ϕ_2 and for $l < k$ $s_l T s_{l+1}$ and s_l is labelled ϕ_1

4. label s by **EG** ϕ_1 if there exists a sequence $s = s_0, s_1, \dots, s_k, \dots, s_m$ of states such that
 - (a) $s_l \rightarrow s_{l+1}$ for all $l < m$
 - (b) $s_k = s_m$
 - (c) all s_l are labelled ϕ_1

State Space Explosion Problem

Complexity $O(|M| \cdot |\phi|)$ looks efficient, since the specification ϕ is written by hand, so probably not too large.

But ... M is usually obtained from n processes operating in parallel. If each has at most k states, we have k^n states for the system as a whole!

Symbolic Model Checking (McMillan, 1990)

An alternate view of the algorithm:

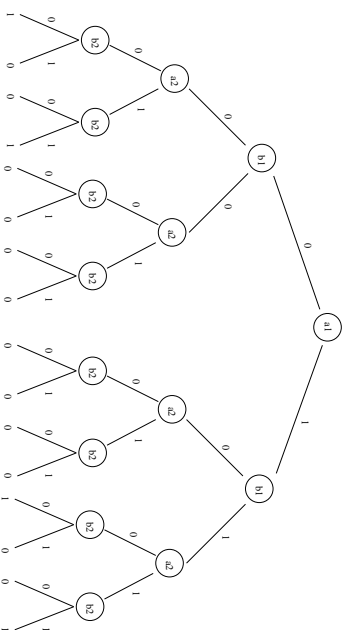
Let $V = \{v_1, \dots, v_n\}$ be a set of boolean variables such that states of M are represented by a function $\pi : V \rightarrow \{0, 1\}$.

For each subformula ψ of ϕ , let S_ψ be the set of states that the algorithm labels by ψ .

Can represent this set by its characteristic function

$f_\psi : \{0, 1\}^n \rightarrow \{0, 1\}$ such that $f_\psi(b_1, \dots, b_n) = 1$ iff the state represented by the assignment $\pi = \{v_1 \mapsto b_1, \dots, v_n \mapsto b_n\}$ is in S_ψ .

(Ordered) Binary Decision Trees



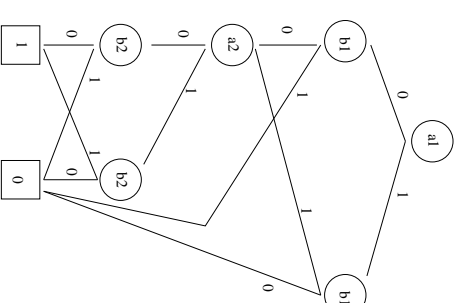
$$f(a1, a2, b1, b2) = (a1 \equiv a2) \wedge (b1 \equiv b2)$$

(Ordered) Binary Decision Diagrams

We can eliminate a lot of redundancies in BDT's by repeatedly applying the following *reduction* rules:

1. identify all 0-leaves, and similarly all 1-leaves
2. if two nodes labelled a have the same 0-successor and the same 1-successor, identify them
3. if a node has the same 0-successor and 1-successor eliminate this node, and direct all incoming edges to this successor

The result, which does not depend on the order of application, is a dag.



Boolean Operations on functions / OBDD's

Example: Let $f, g : \{0, 1\}^k \rightarrow \{0, 1\}$. Then $f \wedge g : \{0, 1\}^k \rightarrow \{0, 1\}$ is defined by

$$(f \wedge g)(b_1, \dots, b_k) = f(b_1, \dots, b_k) \wedge g(b_1, \dots, b_k)$$

Given a boolean operation \star on functions, BDD representation of f and an OBDD representation of g , we can compute an OBDD representation of $f \star g$

Restriction of a function: if $f|_{\{v_1, \dots, v_k\}}$ a boolean function of variables $\{v_1, \dots, v_k\}$ and $b \in \{0, 1\}$, then $f|_{v_i \leftarrow b}$ is a function of variables $\{v_1, \dots, v_k\} \setminus \{v_i\}$,

$$f|_{v_i \leftarrow b}(b_1, \dots, b_{i-1}, b_{i+1}, \dots, b_k) = f(b_1, \dots, b_{i-1}, b, b_{i+1}, \dots, b_k)$$

To compute the OBDD for $f|_{v_i \leftarrow b}$ from the OBDD for f :

1. do a depth first search through the OBDD for f when you reach a node N with a c -successor M ($c \in \{0, 1\}$) labelled v_i , make the c -successor of N be the b -successor of M .
2. reduce the resulting using the rules above

Shannon expansion of a function:

$$f = (\neg v_i \wedge f|_{v_i \leftarrow 0}) \vee (v_i \wedge f|_{v_i \leftarrow 1})$$

Apply this to compute $OBDD(f \star g)$, recursively:

1. if $OBDD(f)$ and $OBDD(g)$ are both terminals with values x, y , $OBDD(f \star g)$ is terminal with value $x \star y$, else

2. Let v_i, v_j be the variables at the roots of the OBDD's for f, g resp

If $i = j$: $OBDD(f \star g)$ has root labelled v_i , and

0-successor $OBDD(f|_{v_i \leftarrow 0} \star g|_{v_i \leftarrow 0})$ and

1-successor $OBDD(f|_{v_i \leftarrow 1} \star g|_{v_i \leftarrow 1})$)

If $i < j$: as for $i = j$, but note g does not depend on v_i , so

$g|_{v_i \leftarrow 0} = g$, similarly for $i > j$.

Proceed recursively, but cache results to avoid recomputing/exponential growth.

Quantified Boolean Logic

Let f be a boolean function of variables V and $v \in V$ a variable

$\exists v(f)$ is a boolean function of variables $V \setminus \{v\}$, defined by

$$\exists v(f) = f|_{v \leftarrow 0} \vee f|_{v \leftarrow 1}$$

$\forall v(f)$ is a boolean function of variables $V \setminus \{v\}$, defined by

$$\forall v(f) = f|_{v \leftarrow 0} \wedge f|_{v \leftarrow 1}$$

Thus, we can compute BDD's for $\exists v(f)$ and $\forall v(f)$ from $OBDD(f)$ using the techniques above.

Applying OBDD's to CTL model checking

Idea: instead of working with individual states, work with sets of states, represented by OBDD's.

Representing the transition relation \rightarrow symbolically:

Let $V = \{v_1, \dots, v_n\}$ be the variables used to represent states.

Define the boolean function f_{\rightarrow} of $2n$ arguments by

$f_{\rightarrow}(b_1, \dots, b_n, b'_1, \dots, b'_n) = 1$ iff $s \rightarrow s'$, where s is the state represented by $\{v_1 \mapsto b_1, \dots, v_n \mapsto b_n\}$ and s' is the state represented by $\{v_1 \mapsto b'_1, \dots, v_n \mapsto b'_n\}$.

We associate with a formula ϕ of CTL, a function f_{ϕ} over the set of variables $V = \{v_1, \dots, v_n\}$ used to represent states of M , such that if the state s is represented by $\{v_1 \mapsto b_1, \dots, v_n \mapsto b_n\}$ then $M, s \models \phi$ iff $f_{\phi}(b_1, \dots, b_n) = 1$.

Computing f_{ϕ} :

1. if $\phi = v_i \in Prop$ then $f_{\phi}(b_1, \dots, b_n) = b_i$
2. $f_{\neg\alpha} = \neg f_{\alpha}$
3. $f_{\alpha \wedge \beta} = f_{\alpha} \wedge f_{\beta}$ (computed as above)

$$f_{EX\alpha} = CheckEX(f_{\alpha})$$

where

$CheckEX(f[v_1, \dots, v_n])$ is

$$\exists v'_1 \dots \exists v'_n (f_{\rightarrow}[v_1, \dots, v_n, v'_1, \dots, v'_n] \wedge f[v'_1, \dots, v'_n])$$

Fixpoints (Tarski-Knaster)

Let $T : \mathcal{P}(S) \rightarrow \mathcal{P}(S)$.

T is *monotonic* if $X \subseteq Y$ implies $T(X) \subseteq T(Y)$ for all $X, Y \subseteq S$.

X is a *fixpoint* of T if $T(X) = X$.

X is a *least fixpoint* of T if X is a fixpoint that is contained in all other fixpoints.

Note that if T is monomtonic then the sequence X_i defined by $X_0 = \emptyset$ and $X_{i+1} = T(X_i)$ satisfies $X_0 \subseteq X_1 \subseteq X_2 \subseteq \dots$

If S is finite, then for some $n \leq |S|$, we have $X_m = X_n$ for all $m \geq n$.

Proposition: If S is finite and T is monotonic, then it has a least fixpoint, which is equal to X_n .

A fixpoint characterization of $\mathbf{E}\phi_1 \cup \phi_2$

Note first $(\mathbf{E}\phi_1 \cup \phi_2) \equiv \phi_2 \vee (\phi_1 \wedge \mathbf{E}\mathbf{X}(\phi_1 \cup \phi_2))$

Given a set $X \subseteq S$ of states of M , define $M[x \mapsto X]$ to be the Kripke structure exactly like M except that the proposition x is true at states in X and nowhere else.

Proposition: Let $T : \mathcal{P}(S) \rightarrow \mathcal{P}(S)$ be defined by

$$T(X) = \{t \mid (M[x \mapsto X], t) \models \phi_2 \vee (\phi_1 \wedge \mathbf{E}\mathbf{X}x)\}.$$

Then

$$\{t \mid (M, t) \models \mathbf{E}\phi_1 \cup \phi_2\}$$

is the least fixpoint of T .

Computing $f_{\mathbf{E}\phi_1 \cup \phi_2}$:

First compute f_{ϕ_1}, f_{ϕ_2} .

Let $f = f_{\text{false}}$

while $f \neq (f_{\phi_2} \vee (f_{\phi_1} \wedge \text{CheckEX}(f)))$

do $f := (f_{\phi_2} \vee (f_{\phi_1} \wedge \text{CheckEX}(f)))$

return(f)

A fixpoint characterization of $\mathbf{E}\mathbf{G}\phi$

$$\mathbf{E}\mathbf{G}\phi \equiv \phi \wedge \mathbf{E}\mathbf{X}(\mathbf{E}\mathbf{G}\phi)$$

In this case, its the *greatest fixpoint*, computed as follows:

First compute f_ϕ .

Let $f = f_{\text{true}}$

while $f \neq (f_\phi \wedge \text{CheckEX}(f))$

do $f := (f_\phi \wedge \text{CheckEX}(f))$

return(f)

Symbolic LTL model checking

Similar ideas can be applied to LTL.

In this case, we represent states, transitions, etc of the Büchi automata symbolically.