Chapter 4

CTL and CTL*

Course "Model checking"

Martin Steffen

Autumn 2021

Section

Introduction

Chapter 4 "CTL and CTL*"
Course "Model checking"
Martin Steffen
Autumn 2021

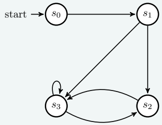
Computation tree logic

- CTL: Computation *tree* logic [2] [3]
- prominent branching time logic
- branching vs. linear time
- remember LTL: models are *paths*, here trees
- we could write

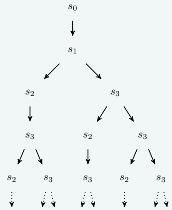
$$s \models \forall\varphi \quad \text{iff} \quad \pi \models \varphi \quad \text{for all path } \pi \text{ starting in } s \tag{1}$$

Unfolding a transition system to a tree

Transition system

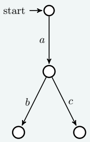


Unfolding

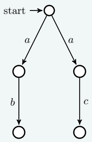


Linear vs. branching

a(b+c)

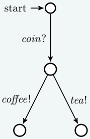


ab+ac

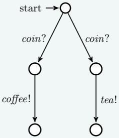


Vending machine example

a(b+c)



ab+ac



Section

Computation tree logic

Chapter 4 "CTL and CTL*"
Course "Model checking"
Martin Steffen
Autumn 2021

Syntax

$$\begin{array}{lcll} \Phi & ::= & \top \mid p \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi \mid \forall\varphi & \text{state formulas} \\ \varphi & ::= & \bigcirc\Phi \mid \Phi_1\ U\ \Phi_2 & \text{path formulas} \end{array}$$

Note: syntactic restriction.

Example: ∞

$$\forall\Box\forall\Diamond green$$

Example: mutex and progress

$$\forall\Box(\neg crit_1 \vee \neg crit_2)$$

$$(\forall\Box\forall\Diamond crit_1) \wedge (\forall\Box\forall\Diamond crit_2)$$

Response

Response

$$\forall\Box(request \rightarrow \forall\Diamond response)$$

Restart

LTL?

Restart

$$\forall\Box\exists\Diamond start$$

LTL?

Derived syntax

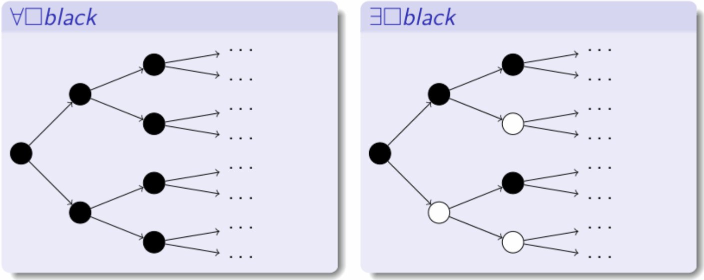
Semantics: $\models$

$$
\begin{array}{lll}
s \models p & \textit{iff} & p \in V(s) \\
s \models \neg\Phi & \textit{iff} & \text{not } s \models \Phi \\
s \models \Phi_1 \wedge \Phi_2 & \textit{iff} & s \models \Phi_1 \text{ and } s \models \Phi_2 \\
s \models \exists\varphi & \textit{iff} & \pi \models \varphi \text{ for some } \pi \in paths(s) \\
s \models \forall\varphi & \textit{iff} & \pi \models \varphi \text{ for all } \pi \in paths(s)
\end{array}
$$

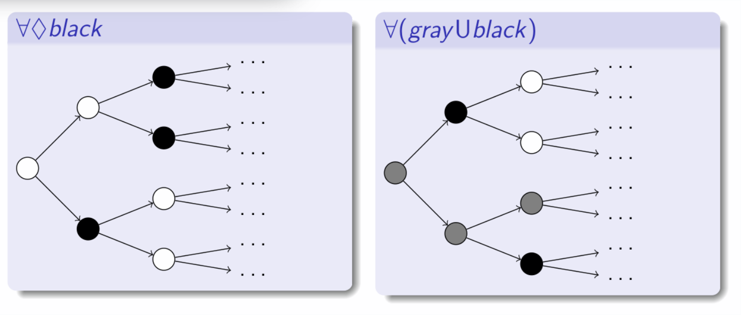
$$
\begin{array}{lll}
\pi \models \bigcirc\Phi & \textit{iff} & \pi^1 \models \Phi \\
\pi \models \Phi_1 \; U \; \Phi_2 & \textit{iff} & \exists j \geq 0.(\pi^j \models \Phi_2 \text{ and } \forall 0 \leq k < j.\pi^k \models \Phi_1)
\end{array}
$$

Semantics for transition systems

CTL semantics, example 1

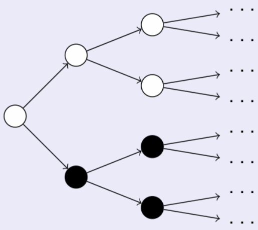


CTL semantics, example 2

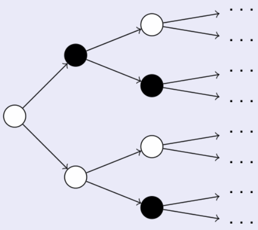


CTL semantics, example 3

$\exists \bigcirc \forall \Box \mathit{black}$



$\forall \Box \exists \bigcirc \mathit{black}$



Semantics: Sat-set

The satisfaction set $Sat(\Phi)$ is the set of states in a transition system TS that satisfies Φ.

A transition system satisfies Φ, written $TS \models \Phi$, iff all the initial states of the TS satisfies Φ: $I \subseteq Sat(\Phi)$, where I is the set of initial states in TS.

Comparison with LTL

CTL and LTL are not equally expressive, but neither is more expressive than the other.

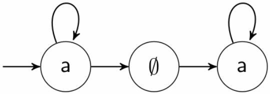
Theorem 6.18 [1]

Let Φ be a CTL formula, and φ the LTL formula that is obtained by eliminating all path quantifiers in Φ. Then:

$$\Phi \equiv \varphi \text{ or there exists no LTL formula that is equivalent to } \Phi.$$

Lemma 6.19

$\forall\Diamond\forall\Box a \not\equiv \Diamond\Box a$



Section

CTL model checking

Chapter 4 “CTL and CTL*”
Course “Model checking”
Martin Steffen
Autumn 2021

Intro

- focus on ENF ($\exists\bigcirc$, $\exists\, U$, $\exists\Box$)
- recursive over structure of formula
- calculate $sat(\Phi)$
- check $I \subseteq sat(\Phi)$
- "global" model checking
- bottom-up traversal of the parse tree of Φ

Normal forms

- state-formulas only
- remember LTL
- positive normal form
- interesting for us: existential NF

ENF

$$
\begin{array}{rcl}
\Phi & ::= & \top \mid p \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \\
 & \mid & \exists \bigcirc \Phi \mid \exists \Phi_1 \; U \; \Phi_2 \mid \exists \Box \Phi \mid
\end{array}
$$

Model checking CTL

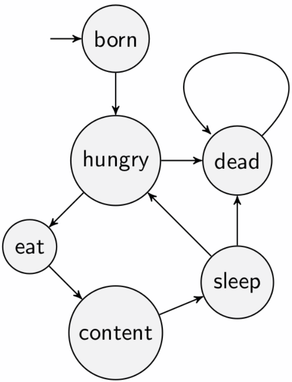
The task is to check whether a transition system TS satisifies a CTL formula Φ. This is the case when all the initial states I of the TS satisfy Φ.

Basic Algorithm

1. The set $Sat(\Phi)$ of all states satisfying Φ is computed recursively ("from inside and out")
2. $TS \models \Phi$ iff $I \subseteq Sat(\Phi)$

This can achieved by a bottom-up traversal of the CTL formula's parse tree.

Model checking: example 1



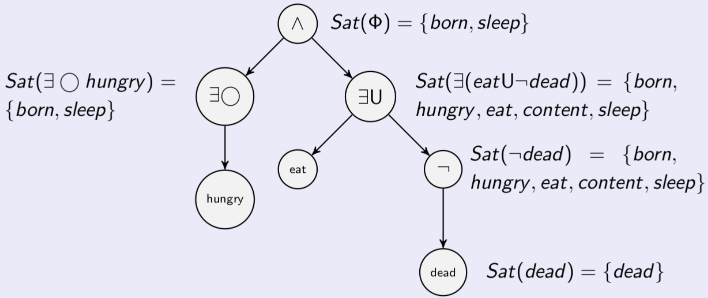
$\forall\Diamond dead$?

$Sat(\forall\Diamond dead) = \{dead\}$

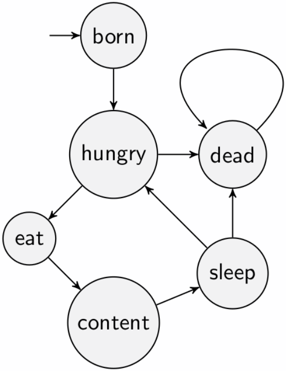
The initial state $born \notin Sat(\forall\Diamond dead)$, so $TS \not\models \forall\Diamond dead$

Model checking: example 2

$\Phi = \exists \bigcirc\, hungry \wedge \exists(eat \; \mathsf{U} \; \neg dead)$



Model checking: example 2



$$Sat(\Phi) = \{born, sleep\}$$

Because the only initial state is in the formula's satisfaction set, the transition system satisfies the formula.

Basic algorithm

```
input:  finite transition system 𝒯 and Φ
output: 𝒯 ⊨ Φ
________________________________________________
for all i ≤ |Φ| do
   for all Ψ ∈ sub(Φ) with |Ψ| = i do
      compute sat(Ψ) from sat(Ψ′) (* max. genuine Ψ′ ⊆ sat(Ψ) *)
   od
od
return I ⊆ sat(Φ)
```

Recursive algorithm

```
input:  finite transition system 𝒯 and Φ
output: 𝒯 ⊨ Φ
-----------------------------------------------------------
switch Φ
     ⊤              => return S
     p              => return {s ∈ S | p ∈ V(s)}
     Φ₁ ∧ Φ₂        => return sat(Φ₁) ∩ sat(Φ₂)
     ∃◯Φ            => return {s ∈ S | post(s) ∩ sat(Φ)}
     ∃(Φ₁ U Φ₂)     => ``lfp for ∃U''
     ∃□Φ            => ``gfp for ∃□''
```

Backward calculation of $\exists\Diamond$

```
T := sat(B);
while pre(T) \ T ≠ ∅ do
   let s ∈ pre(T) \ T ;
   T := T ∪ {s};
od;
return T;
```

Massaging the algo

```
T := sat(B);
while pre(T) \ T ≠ ∅ do
  let s ∈ pre(T) \ T ;
  T := T ∪ {s};
od;
return T;
```

loop body

$$T := T \cup \{pick(pre(T))\}$$

Loop condition

$$T \supset T \cup \{pick(pre(T))\} \quad \text{or} \quad T \neq T \cup \{pick(pre(T))\} .$$

Massaging the algo

```
T := sat(A);
while  T ≠ T ∪ {pick(pre(T))}
   T := T ∪ pick(pre(T));
od;
return T;
```

loop body

$$T := T \cup \{pick(pre(T))\}$$

Loop condition

$$T \supset T \cup \{pick(pre(T))\} \quad \text{or} \quad T \neq T \cup \{pick(pre(T))\} \; .$$

Iteration

exit condition

$$T = T \cup \{pick(pre(T))\} \ .$$

$$\begin{array}{rcl} T_0 & = & A \\ T_{j+1} & = & F(T_j) \quad \text{where} \quad F(X) = pick_{\emptyset}(pre(X)) \cup X \end{array}$$

Stabilization

$$A = T_0 \subset T_1 \subset T_2 \subset \ldots \subset T_k = T_{k+1} = T_{k+2} \ldots$$

Characterization

Goal (a)

Find me a set T such that (a) it contains A and such that (b) $F(T)$ does not make it larger.

Characterization

Goal (a)

Find me a set T such that (a) it contains A and such that (b) $F(T)$ does not make it larger.

Goal (a)

$$\begin{array}{lll} T & \supseteq & A \\ T & \supseteq & F(T) \qquad \text{where} \quad F(X) = pick_{\emptyset}(pre(X)) \cup X \end{array}$$

Characterization

Goal (a)

Find me a set T such that (a) it contains A and such that (b) $F(T)$ does not make it larger.

Goal (a)

$$
\begin{array}{lcll}
T & \supseteq & A & \\
T & \supseteq & F(T) & \quad \text{where} \quad F(X) = pick_{\emptyset}(pre(X)) \cup X
\end{array}
$$

$$
T \supseteq F'(T) \quad \text{where} \quad F'(X) = pick_{\emptyset}(pre(X)) \cup X \cup A
$$

Fixpoint

$$A = T_0 \subset T_1 \subset T_2 \subset \ldots \subset T_k = T_{k+1} = T_{k+2} \ldots$$

Fixpoint

$$T_{k+1} = F(T_k) = T_k$$

(Pre-)Fixpoint

Goal (a)

Find me a set T such that (a) it contains A and such that (b) $F(T)$ does not make it larger.

T_k solves the following

fixpoint

$$F(X) = X$$

(Pre-)Fixpoint

Goal (a)

Find me a set T such that (a) it contains A and such that (b) $F(T)$ does not make it larger.

T_k solves the following

fixpoint	pre-fixpoint
$F(X) = X$	$F(X) \subseteq X$

That was only half of the characterization

Goal (a)

Find me a set T such that (a) it contains A and such that (b) $F(T)$ does not make it larger.

$$A = T_0 \subset T_1 \subset T_2 \subset \ldots \subset T_k = T_{k+1} = T_{k+2} \ldots$$

That was only half of the characterization

Goal (a)

Find me a set T such that (a) it contains A and such that (b) $F(T)$ does not make it larger.

$$A = T_0 \subset T_1 \subset T_2 \subset \ldots \subset T_k = T_{k+1} = T_{k+2} \ldots$$

Goal (b)

Find the **smallest** set T satisfying **goal (a)**.

Smallest (pre-)fixpoint

- interested in the *smallest* (pre)-fixpoint

fixpoint	**pre-fixpoint**
$F(X) = X$	$F(X) \subseteq X$

Facts

1. unique
2. smallest fixpoint = smallest pre-fixpoint
3. T_k in the iteration is actually that lfp

Explicit-state vs. symbolic model checking

The code again:

```
T := sat(A);
while T ≠ T ∪ {pick(pre(T))}
  T := T ∪ pick(pre(T));
od;
return T;
```

Explicit-state	**symbolic**
exploring states individually	exploring sets of states (sat)

Breadth-first?

```
T := sat(B);
while T ≠ T ∪ pre(T) do
  T := T ∪ pre(T)
od;
return T;
```

- likewise: fix-point

But is it actually better?

Breadth-first?

```
T := sat(B);
while T ≠ T ∪ pre(T) do
   T := T ∪ pre(T)
od;
return T;
```

- likewise: fix-point

But is it actually better?

Not really

if the exploration adds $pre(T)$ states **individually.**

Breadth-first?

```
T := sat(B);
while T ≠ T ∪ pre(T) do
   T := T ∪ pre(T)
od;
return T;
```

- likewise: fix-point

But is it actually better?

Not really

if the exploration adds $pre(T)$ states **individually.**

better

if one can calculate $pre(T)$ *all at once!*

Symbolic model checking

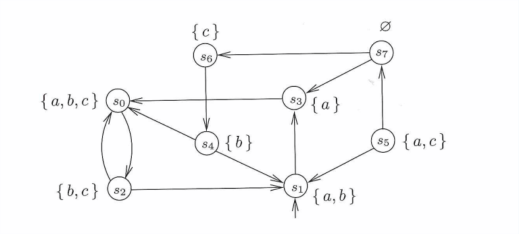
key: efficient representation of

- transition system
- sets of states
- different operations on those sets, in particular
- symbolic exploration by efficent calculation of pre in a set of state

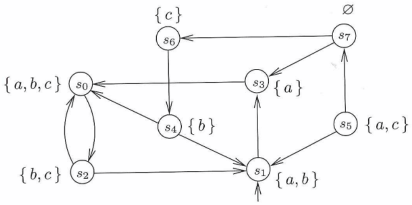
First: characterize all operators

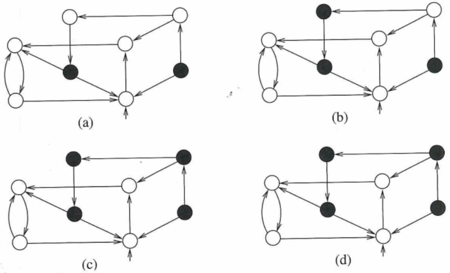
1. $sat(\top) = S$.
2. $sat(p) = \{s \in S \mid p \in V(s), \text{ for any } p \in P\}$.
3. $sat(\Phi_1 \wedge \Phi_2) = sat(\Phi_1) \cap sat(\Phi_2)$.
4. $sat(\neg\Phi) = S \setminus sat(\Phi)$.
5. $sat(\exists \bigcirc \Phi) = \{s \in S \mid \exists s'.s \to s' \wedge s' \in sat(\Phi)\}$
6. $sat(\exists(\Phi_1 \; U \; \Phi_2))$ is the *smallest* subset T of S such that
   - 6.1 $sat(\Phi_2) \subseteq T$ and
   - 6.2 $s \in sat(\Phi_1)$ and $\exists s' \in T.s \to s'$ implies $s \in T$.
7. $sat(\exists\Box\Phi)$ is the *largest* subset T of S such that
   - 7.1 $T \subseteq sat(\Phi)$ and
   - 7.2 $s \in T$ implies $\exists s' \in T.s' \to s$.

Example $\exists(\top \ U \ (a = c) \wedge (a \neq b))$



Example $\exists(\top\ U\ (a = c) \wedge (a \neq b))$





Section

Symbolic model checking

Chapter 4 “CTL and CTL*”
Course “Model checking”
Martin Steffen
Autumn 2021

Symbolic?

"Symbolic Model Checking: 10^{20} States and beyond" [1]

- explicit state vs. symbolic
- normal forms

Switching functions

"bit-vectors"

$$Eval(Var) = Var \to \{0,1\} \ .$$

Switching function

$$f : Eval(Var) \to \{0,1\} = (Var \to \{0,1\}) \to \{0,1\}$$

cf. propositional semantics $[\![\varphi]\!]$

Boolean operators on switching functions

- cf. *truth tables*
- all boolean operators and constants have (of course) an analogue on switching functions
- syntax vs. semantics
- canonical (and/or normal) forms

Operators on switching functions (2)

Projection of a "bit-vector" onto a variable

$$proj_{z_i} : (Var \to \{0,1\}) \to \{0,1\}$$

Disjunction

Operators on switching functions (2)

Projection of a "bit-vector" onto a variable

$$proj_{z_i} : (Var \to \{0,1\}) \to \{0,1\}$$

Disjunction

$$(f_1 \vee f_2)([z_1 = b_1, \ldots, z_k = b_k]) = \max(f_1([z_1 = b_1, \ldots, z_m = b_m]), f_2([z_n = b_n, \ldots, z_k = b_k]))$$

Cofactors

Definition (Cofactors)

Assume a variable set $Var = \{z, y_1, \ldots, y_m\}$ and let $f : (Var \to \{0,1\}) \to \{0,1\}$ be a switching function over it. The *positive cofactor* of f for variable z, written $f\mid_{z=1}$ is the switching function given by

$$f\mid_{z=1}(b_1, \ldots, b_m) = f(1, b_1, \ldots, b_m) \quad (2)$$

The *negative cofactor* of f for z, written $f\mid_{z=0}$ is defined analogously. If f is a switching function for $\{z_1, \ldots, z_k, y_1, \ldots, y_m\}$, then we write $f\mid_{z_1=b_1,\ldots,z_k=b_k}$ for the *iterated cofactor* of f, given by

$$f\mid_{z_1=b_1,\ldots,z_k=b_k} = (\ldots(f\mid_{z=b_1})\ldots)\mid_{z_k=b_k} \quad (3)$$

Essential variables

Definition (Essential variable)

A variable z is *essential* for a switching function, if

$$f\ |_{z=1} \neq f\ |_{z=0} \quad . \tag{4}$$

Shannon expansion

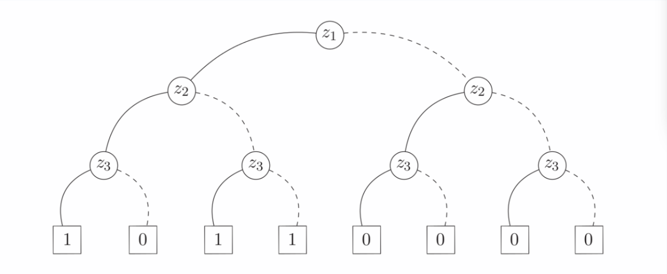
Lemma (Shannon expansion)

Let f be a switching function for Var. Then

$$f = (\neg z \wedge f\ |_{z=1}) \vee (z \wedge f\ |_{z=0}) \tag{5}$$

for all variables z from Var.

Binary decision tree $z_1 \wedge (\neg z_2 \vee z_3)$



Propositional encoding

$$\mathcal{T} = (S, \rightarrow, I, P, L)$$

Task: encode by boolean formulas / switching functions

- all ingredients of $\mathcal{T}$
- $sat(\_)$
- realise operations during the mc-algo by operations on the encodings, in particular pre

Characteristic function

Equivalent (isomorphic) views

- $2^S \equiv S \to \{0,1\}$ (or $S \to \mathbb{B}$)
- $B \subseteq S \qquad \chi_B$

Encoding states and subsets of states

- that this is possible, should be obvious
- use propositional variables $x_1, \ldots, x_n$
- padding: assume $S = Eval(\vec{x}) = \{x_1, \ldots, x_n\} \to \{0, 1\}$

Sets of states $B \subseteq S$

$$\chi_B : (Eval(\vec{x}) \to \{0,1\} = (\vec{x} \to \{0,1\}) \to \{0,1\}$$

Encoding the transition relation $\rightarrow$

$$\rightarrow \subseteq \; S \times S$$

- make a "copy" of $\vec{x}$: different variables $\vec{x}'$
- renaming operation on switching functions $[y \leftarrow x]$

Encode $\rightarrow$ as Δ

$$\Delta : ((\vec{x}, \vec{x}') \rightarrow \{0,1\}) \rightarrow \{0,1\},$$

$$\Delta(s_1, s_2[\vec{x}' \leftarrow \vec{x}]) = \begin{cases} 1 & \text{if } s_1 \rightarrow s_2 \\ 0 & \text{else} \end{cases}$$

Remember charaterization and the fixpoints

1. $sat(\top) = S$.
2. $sat(p) = \{s \in S \mid p \in V(s),$ for any $p \in P\}$.
3. $sat(\Phi_1 \wedge \Phi_2) = sat(\Phi_1) \cap sat(\Phi_2)$.
4. $sat(\neg\Phi) = S \setminus sat(\Phi)$.
5. $sat(\exists \bigcirc \Phi) = \{s \in S \mid \exists s'.s \to s' \wedge s' \in sat(\Phi)\}$
6. $sat(\exists(\Phi_1 \; U \; \Phi_2))$ is the *smallest* subset T of S such that
   - 6.1 $sat(\Phi_2) \subseteq T$ and
   - 6.2 $s \in sat(\Phi_1)$ and $\exists s' \in T.s \to s'$ implies $s \in T$.
7. $sat(\exists\Box\Phi)$ is the *largest* subset T of S such that
   - 7.1 $T \subseteq sat(\Phi)$ and
   - 7.2 $s \in T$ implies $\exists s' \in T.s' \to s$.

$\exists \bigcirc A$ means, calculating pre

pre-calulation as switching function

$$\exists \vec{x}'. \underbrace{\Delta(\vec{x}, \vec{x}')}_{s \in pre(s')} \wedge \underbrace{\chi_A[\vec{x}' \leftarrow \vec{x}]}_{s' \in A}$$

$\exists\Diamond$: basically iteration over pre

One step: $T_{j+1} = pre(T_j)$

$$\exists \vec{x}'.\, \underbrace{\Delta(\vec{x}, \vec{x}')}_{s \in pre(s')} \wedge \underbrace{f_j[\vec{x}' \leftarrow \vec{x}]}_{s' \in T_j} \tag{6}$$

$\exists\, U$: not much harder...

```
f_0(x⃗)  :=  χ_{A_1}(x⃗);
j  :=  0;
repeat
   f_{j+1}(x⃗)  :=  f_{j+1}(x⃗) ∨ (χ_{A_2}(x⃗) ∧ ∃x⃗'.Δ(x⃗, x⃗') ∧ f_j(x⃗'));
until  f_j(x⃗) = f_{j-1}(x⃗);
return   f_j(x⃗).
```

finally $\exists\Box$

$f_0(\vec{x}) := \chi_A(\vec{x})$;
$j := 0$;
repeat
  $f_{j+1}(\vec{x}) := f_{j+1}(\vec{x}) \wedge \exists \vec{x}'.\Delta(\vec{x}, \vec{x}') \wedge f_j(\vec{x}'))$;
until $f_j(\vec{x}) = f_{j-1}(\vec{x})$;
return $f_j(\vec{x})$.

Largest fixpoint

- sets get smaller in the iteration

Section

Binary decision diagrams

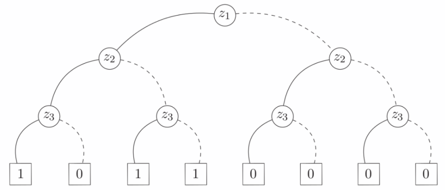
Chapter 4 “CTL and CTL*”
Course “Model checking”
Martin Steffen
Autumn 2021

How to efficiently implement all that?

- switching functions

$$f : Eval(Var) \to \{0,1\} = (Var \to \{0,1\}) \to \{0,1\}$$

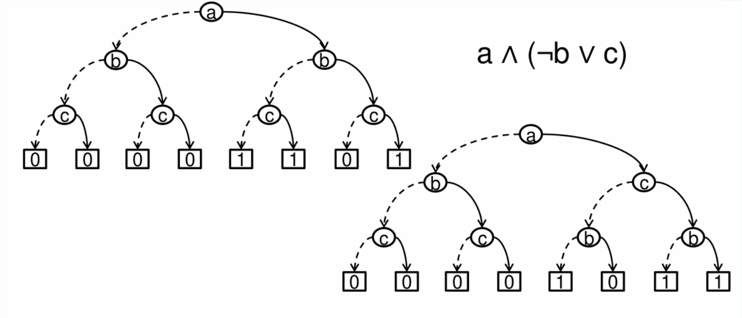
- $+$ operations thereon
- *binary decision trees*



$$z_1 \wedge (\neg z_2 \vee z_3)$$

Issues

- size: still exponential
- non-canonical



From BDTs to (RO)BDDs

- addressing both problems
- *reduced ordered binary decision diagrams*
- often "BDDs" just mean ROBDDs
- two general ideas

Two general ideas

to addess both mentioned problems.

Canonicity: fix an order on the variables

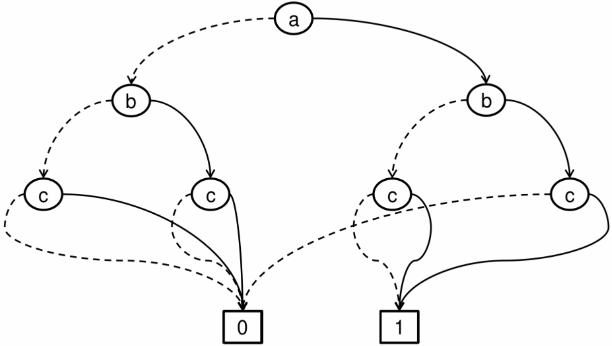
Size: don't represent duplicate parts of the graph more than once

As first easy step

BDT to OBDD

- have only 2 terminal nodes (for 0 and for 1), no duplicate leaves (BDD), and
- fix an order on the var's (OBDD)

Example OBDD



$a \wedge (\neg b \vee c)$ with order $a < b < c$

Reduced OBBDs

Reduce

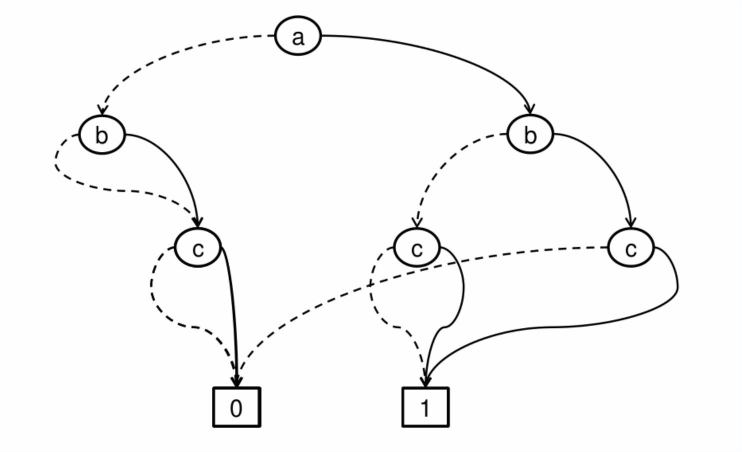
Uniqueness

no 2 nodes for the same variable have the "same" high- and low-children $\Rightarrow$ merge isomorphic subgraphs

non-redundent tests

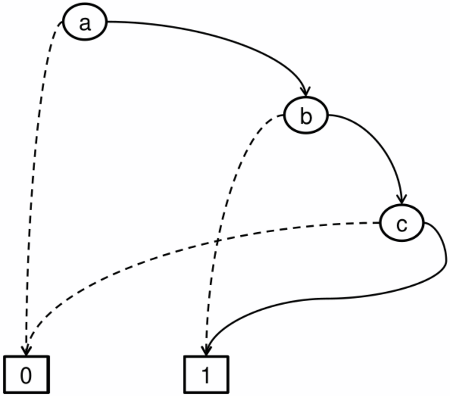
no variable node has identical high and low children

Merge isomorphic subgraphs



$a \wedge (\neg b \vee c)$ with order $a < b < c$

Remove reduncancy



$a \wedge (\neg b \vee c)$ with order $a < b < c$

ROBDDs as canonical representation

Canonicity

For every boolean function $f : (Var \to \{0,1\}) \to \{0,1\}$ and a give variable ordering, there exists exactly one ROBDD representing f

facts

equivalence checking

satisfiability

ROBDDs as canonical representation

Canonicity

For every boolean function $f : (Var \rightarrow \{0,1\}) \rightarrow \{0,1\}$ and a give variable ordering, there exists exactly one ROBDD representing f

facts

equivalence checking

linear time

satisfiability

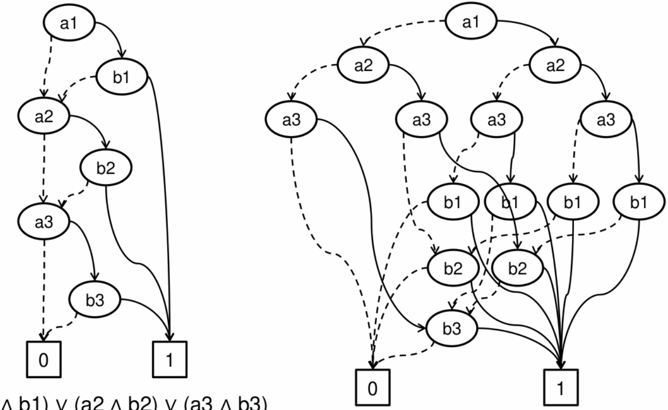
constant time

But where is the catch?

Satisfiability

Isn't SAT NP-complete?

Sensitivity to variable order



(a1 ∧ b1) ∨ (a2 ∧ b2) ∨ (a3 ∧ b3)

Sensitivity to variable order

- different variable orders $\Rightarrow$ different ROBBDs
- crucial in practice to find a (in many cases) good order
- finding the best: NP-hard
- heuritics exists

Representing boolean functions is not all

- canonical, often (but not always) compact representation
- we also need to *"work"* with them
- remember the CTL model checking algorithms

Boolean operators (Apply)

- boolean operators on (R)OBBDs
- *recursively* over the two OBDDs
- based on Shannon's (or Boole's) expansion
- preserve the order
- if working on ROBBs, re-reduce the result.

Logical operations on OBDDs

Logical negation ($\neg$)

- Replace the value of each leaf node by its negation

All 16 logical operations can be applied on boolean functions using the Apply algorithm.

Restriction of the variable x_i to a constant b:

- $f|_{x_i \leftarrow b}(x_1, \ldots x_n) = f(x_1, x_2, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_n)$
- $f|_{x_i \leftarrow 1}$: positive Shannon cofactor of f for x_i
- $f|_{x_i \leftarrow 0}$: negative Shannon cofactor of f for x_i
- To compute the new OBDD:
  - We traverse the tree in a depth-first manner
  - All incoming edges to v, s.t. $var(v) = x_i$ should be redirected to $low(v)$ if $b = 0$ or $high(v)$ if $b = 1$
  - Reduce the OBDD

Apply

Shannon expansion:

- $f = (\neg x \wedge f|_{x \leftarrow 0}) \vee (x \wedge f|_{x \leftarrow 1})$
- Allows us to split a problem into two subproblems

Using the *Apply* algorithm to solve all 16 logical operations. Let

- $\bullet$ be a two-argument logical operation (and, or, xor etc.)
- f and f' be two boolean functions
- v and v' be the OBDDs roots for f and f'
- $var(v) = x$ and $var(v') = x'$

If both v and v' are drains:

- $f \bullet f' = val(v) \bullet val(v')$

Apply (cont'd)

If $x = x'$:

- Recursively solve the two subproblems:
  $f \bullet f' = (\neg x \wedge (f|_{x \leftarrow 0} \bullet f'|_{x \leftarrow 0})) \vee (x \wedge (f|_{x \leftarrow 1} \bullet f'|_{x \leftarrow 1}))$
- The root of this new OBDD will be a new node w such that
  - $var(w) = x$
  - $low(w)$ will be OBDD for $(f|_{x \leftarrow 0} \bullet f'|_{x \leftarrow 0})$
  - $high(w)$ will be OBDD for $(f|_{x \leftarrow 1} \bullet f'|_{x \leftarrow 1})$

If $x < x'$ ($x = x_i$ and $x' = x_j$ where $i < j$):

- $f \bullet f' = (\neg x \wedge (f|_{x \leftarrow 0} \bullet f')) \vee (x \wedge (f|_{x \leftarrow 1} \bullet f'))$
- Similar for $x > x'$

Algorithm is polynomial with dynamic programming

Boolean quantification

If f is a function, x is a variable, then

- $\exists x.f = (f|_{x \leftarrow 0}) \vee (f|_{x \leftarrow 1})$
- $\forall x.f = (f|_{x \leftarrow 0}) \wedge (f|_{x \leftarrow 1})$

We need to compute the OBDD for both subproblems using the *Restrict* algorithm:

- $f|_{x \leftarrow 0}$: For each node v where $var(v) = x$
  - Incoming edges are redirected to $low(v)$
  - Remove node v
- $f_{x \leftarrow 1}$: For each node v where $var(v) = x$
  - Incoming edges are redirected to $high(v)$
  - Remove node v

References I

Bibliography

[1] Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L., and Hwang, L. (1992). Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170.

[2] Clarke, E. M. and Emerson, E. A. (1982). Design and synthesis of synchronisation skeletons using branching time temporal logic specifications. In Kozen, D., editor, *Proceedings of the Workshop on Logic of Programs 1981*, volume 131 of *Lecture Notes in Computer Science*, pages 244–263. Springer Verlag.

[3] Queille, J. P. and Sifakis, J. (1982). Specification and verification of concurrent systems in CESAR. In Dezani-Ciancaglini, M. and Montanari, U., editors, *Proceedings of the 5th International Symposium on Programming 1981*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer Verlag.