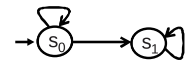
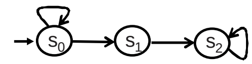
Algorithmic Verification

Comp4151
Lecture 5-A
Ansgar Fehnker

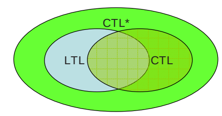
Overview

Modelling
- Finite automata
- Büchi automata
- Kripke structures





Specification
- Linear Time Logic
- Computation Tree Logic
- CTL*



Overview

Explicit state CTL
- Bottom-up recursive labelling algorithm
- Linear in the size of Kripke structure and formula

However
- Suffers from state explosion problem

Fixpoint characterization of CTL
- Leads to set based algorithm

Overview

Semantic of CTL with fixpoints

Given Kripke structure $M=(S, s_0, \rightarrow, \mu)$, with n states, and ϕ in ENF over atomic propositions AP.

- $[|true|] = S$
- $[|false|] = \emptyset$
- $[|\ p\ |] = \{ s \mid p \in \mu(s) \}$
- $[|\neg \phi\ |] = S \setminus [|\ \phi\ |]$
- $[|\ \phi \wedge \psi\ |] = [|\ \phi\ |] \cap [|\ \psi\ |]$
- $[|\ EX\ \phi\ |] = \{ s \mid \exists (s,s') \in \rightarrow s$
- $[|\ E\ \phi\ U\ \psi\ |] = f_{EU}{}^{n}(\emptyset)$
- $[|\ EG\ \phi\ |] = f_{EG}{}^{n}(S)$

- Translates to an algorithm based on sets
- Sets of states as unordered list lists are inefficient
- We need
  - Compact set representation
  - Efficient operations on sets

Binary Decision Diagrams

A longer diversion

With examples and illustrations by Andreas Jacobs and Massimo Benerecetti

Boolean Functions

Definition

A *boolean variable* is variable over values {0,1}.

A *boolean function* over n arguments is a function
f: $\{0,1\}^n \rightarrow \{0,1\}$

Two common representations of boolean functions
- propositional formula
- truth table

Boolean Function

Example:

$f(x_1, x_2, x_3) = (x_1 \vee x_2) \wedge x_3$

compact, but satisfiablity and comparison hard

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	0	0	0
1	0	1	1

exponential size: 2^n lines for n variables

Boolean Function

Binary Decision Tree

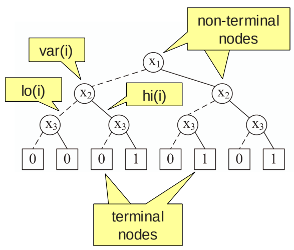
- Represents boolean function as a decision tree
- Non-terminal node i labeled with
  - variable var(i)
  - successors lo(i) and hi(i)
- Terminal nodes are labeled 0 or 1
- Fast lookup of $f(x_1, \ldots, x_n)$ given $x_1, \ldots, x_n$

Boolean Functions

Binary Decision Tree

$f(x_1, x_2, x_3) = (x_1 \vee x_2) \wedge x_3$

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	0	0	0
1	0	1	1

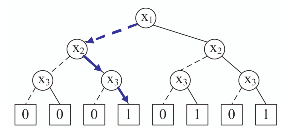


Boolean Functions

Binary Decision Tree

$f(0, 1, 1) = (0 \vee 1) \wedge 1$

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	0	0	0
1	0	1	1

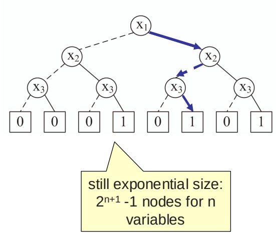


Boolean Functions

Binary Decision Tree

$f(1, 0, 1) = (1 \vee 0) \wedge 1$

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	0	0	0
1	0	1	1



Boolean Functions

Binary Decision Diagrams

Represents boolean functions as *directed acyclic graph* with a single *initial node*

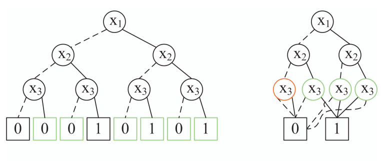
More compact representation than decision trees

Omit duplicated nodes and unnecessary tests.

Binary Decision Diagrams

Reduction

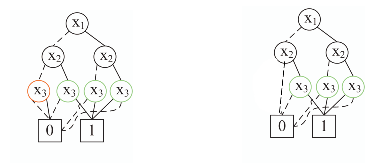
Rule 1: Share identical terminal nodes.



Binary Decision Diagrams

Reduction

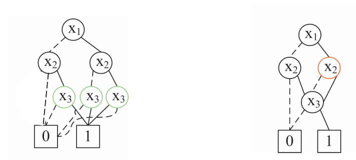
Rule 2: Remove redundant tests



Binary Decision Diagrams

Reduction

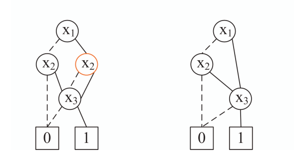
Rule 3: Share identical non-terminal nodes.



Binary Decision Diagrams

Reduction

Rules applied iteratively



Binary Decision Diagrams

Definitions

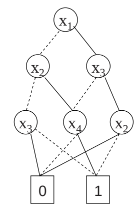
A BDD is *reduced* if none of reduction rules can be applied

Given a variable order $x_1 < \ldots < x_n$ a BDD is *ordered* if for each node i holds

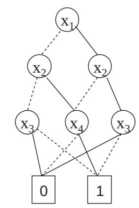
$$j \in \{ lo(i), hi(i) \} \text{ implies } x_i < x_j$$

Binary Decision Diagrams

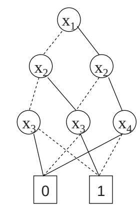
Examples



BDD



OBDD



ROBDD

OBDDs

Theorem

Two reduced ordered BDDs with respect to the same order $x_1 < \ldots < x_n$ represent the same boolean function iff they have the same structure.

Furthermore
- A function is a *tautology* if its ROBDD *u* is *equal* to 1.
- A function is a *satisfiable* if its ROBDD *u* is *not equal* to 0

ROBDDs provide a canonical for boolean functions

OBDDs

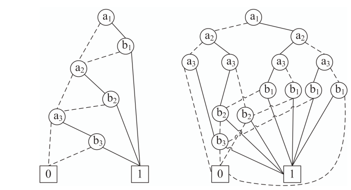
Variable ordering

- Size of an (reduced) OBDD depends on the order

OBDDs

Variable ordering

Example: $(a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee (a_3 \wedge b_3)$



$a_1 < b_1 < a_2 < b_2 < a_3 < b_3$ $\quad$ $a_1 < a_2 < a_3 < b_1 < b_2 < b_3$

OBDDs

Variable ordering

- Size of an (reduced) OBDD depends on the order
- Checking a variable order for optimality is NP-hard
- For some formulas size is exponential for any order
- In practice these cases rarely occur
- There are good heuristics to find good orders

OBDDs

Intermediate Summary

OBDDs provide a compact representation for boolean functions (most of the times)

Satisfiability and comparison are easy to check

However:
- What about other operations on OBDDs?
- What is the relation with model checking?

Operations on OBDDs

Reduce

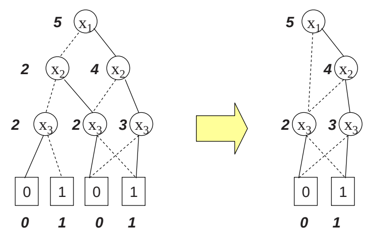
Bottom-up labelling based on the reduction rules

Label nodes with integer numbers as follows
- two terminal nodes with the same value get the same label
- if lo(i)=hi(i) replace set label(i) to label(lo(i))
- if var(i)=var(j) and lo(i)=lo(j) and hi(i)=hi(i) for some j set label(i) to label(j)
- otherwise label(i) with next unused integer

After labelling redirect edges bottom-up

Operations on OBDDs

Reduce



Operations on OBDDs

Restrict

Restrict variable x of function f to constant value b,
i.e. replace all occurrences of x by b.

Example

$f(x_1, x_2, x_3) \quad = (x_1 \vee x_2) \wedge x_3$
$f[0/x_2]\ (x_1, x_2, x_3) \quad = (x_1 \vee 0\ ) \wedge x_3 \ = x_1 \wedge x_3$
$f[1/x_2]\ (x_1, x_2, x_3) \quad = (x_1 \vee 1\ ) \wedge x_3 \ = x_3$

Operations on OBDDs

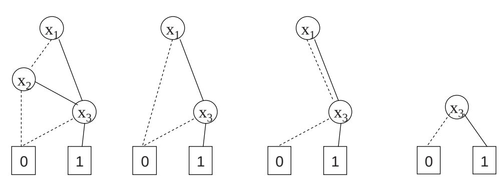
Restrict

Restrict variable x of function f to constant value b,
i.e. replace all occurrences of x by b.

For any node i pointing to a node j with var(j)=x
- repalce pointer of i by lo(j) if b=0
- replace pointer of j by hi(j) if b=1

Operations on OBDDs

Restrict



f $\quad$ $f[0/x_2]$ $\quad$ $f[1/x_2]$ $\quad$ reduce($f[1/x_2]$)

Operations on OBDDs

Apply

$\circ$ can be any boolean operation: and, or, xor, …

Given OBDDs B_f and B_g for boolean formulas f and g.
Apply($\circ$, B_f, B_g) computes the OBDD of $f \circ g$

Basic Idea

- Let x be smallest variable of B_f and B_g (i.e root in B_f or B_g)
- Split into two smaller sub-problem for x=0 and x=1.
- Repeat until leaves are reached. Apply $\circ$ to the leaves.

Recursive application of *Shannon Expansion*

$$f = (\neg x_i \wedge f[0/x_i]) \vee (x_i \wedge f[1/x_i])$$

Operations on OBDDs

Apply

Given OBDDs B_f and B_g for boolean formulas f and g.
Apply($\circ$, B_f, B_g) computes the OBDD of $f \circ g$

Basic Idea

- Let x be smallest variable of B_f and B_g (i.e root in B_f or B_g)
- Split into two smaller sub-problem for x=0 and x=1.
- Repeat until leaves are reached. Apply $\circ$ to the leaves.

Recursive application of *Shannon Expansion*

$$f \circ g = (\neg x_i \wedge (f[0/x_i] \circ g[0/x_i])) \vee (x_i \wedge (f[1/x_i] \circ g[1/x_i]))$$

Operations on OBDDs

Apply

Given OBDDs B_f and B_g for boolean formulas f and g.
Let i_f and i_g be the root nodes of B_f and B_g.

Rule 1:

if i_f and i_g are non-terminal nodes and $var(i_f) = var(i_g)$
$\Rightarrow$ label current node with $var(i_f)$
create a low edge to apply($\circ$,lo(i_f),lo(i_g)) — $f[0/x_i] \circ g[0/x_i]$
create a hi edge to apply($\circ$,hi(i_f),hi(i_g)) — $f[1/x_i] \circ g[1/x_i]$

Operations on OBDDs

Apply

Given OBDDs B_f and B_g for boolean formulas f and g.
Let i_f and i_g be the root nodes of B_f and B_g.

Rule 2:

if i_f is a non-terminal and $var(i_f) < var(i_g)$ or i_g a terminal
$\Rightarrow$ label current node with $var(i_f)$
create a low edge to apply($\circ$,lo(i_f),i_g) — $f[0/x] \circ g$
create a hi edge to apply($\circ$,hi(i_f),i_g) — $f[1/x] \circ g$

$var(i_f) < var(i_g)$ implies $x = var(i_f)$ is not in g, thus $g[0/x] = g$

Operations on OBDDs

Apply

Given OBDDs B_f and B_g for boolean formulas f and g.
Let i_f and i_g be the root nodes of B_f and B_g.

Rule 3:

- if i_f is a non-terminal and $var(i_f) > var(i_g)$ or i_g a terminal
- $\Rightarrow$ Similar to Rule 2

Operations on OBDDs

Apply

Given OBDDs B_f and B_g for boolean formulas f and g.
Let i_f and i_g be the root nodes of B_f and B_g.

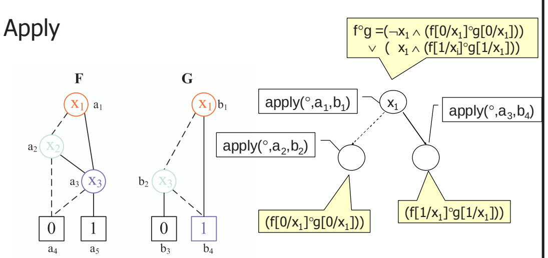
Rule 4:

- if i_f and i_g are terminal nodes labelled b_f and b_g
- $\Rightarrow$ label current node with $b_f \circ b_g$

0 or 1

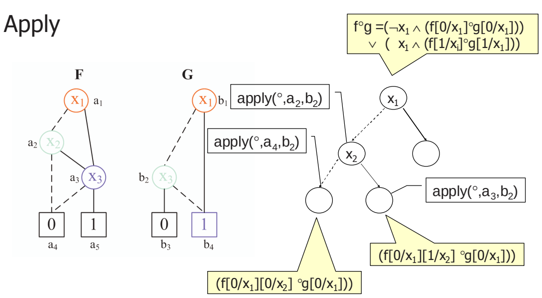
Operations on OBDDs

Apply



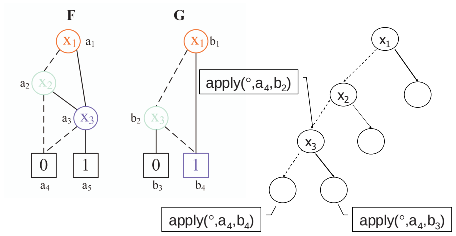
Operations on OBDDs

Apply



Operations on OBDDs

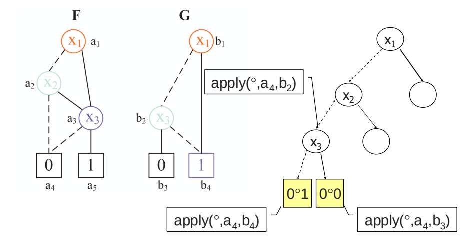
Apply



Comp4151 Ansgar Fehnker

Operations on OBDDs

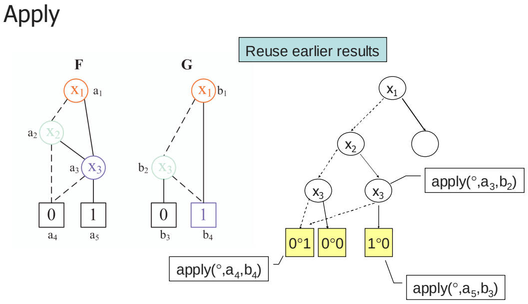
Apply



Comp4151 Ansgar Fehnker

Operations on OBDDs

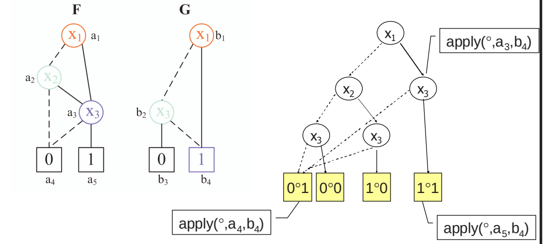
Apply



Comp4151 Ansgar Fehnker

Operations on OBDDs

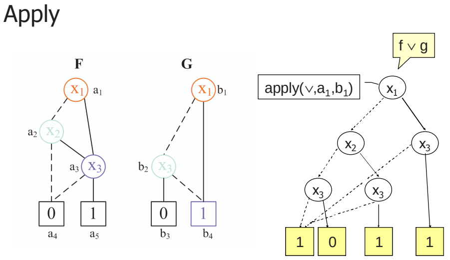
Apply



Comp4151 Ansgar Fehnker

Operations on OBDDs

Apply

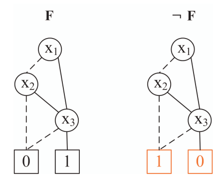


Reduce to obtain ROBDD

Operations on OBDDs

Negate

Given boolean functions f and reduced OBDD B_f, compute ¬f, by swapping the leaves.



Operations on OBDDs

Exist

Given boolean functions f variable x and reduced OBDD B_f, compute ∃x.f.

We have

$$\exists x.f = f[0/x] \vee f[1/x]$$

⇒ We can use an combination of *Restrict* and *Apply*

More efficiency by exploiting common structure in f[0/x] and f[1/x]

Operations on OBDDs

Some complexities

Operation	Complexity
Reduce	O(F)
Negate	constant
Apply	O(|F|*|G|)
Exists	$O(\|F\|*\|G\|*2^{2n})$

Model Checking and OBDDs

Another Intermediate Summary

- OBDDs provide a compact representation for boolean functions (most of the times)
- Satisfiability and comparison are easy to check
- Efficient algorithms for boolean operations

- However:
  - What is the relation with model checking?

Symbolic CTL Model Checking

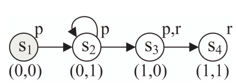
Observations

- *Boolean function* f: $\{0,1\}^n \rightarrow \{0,1\}$ can be used to represent *subsets* of $\{0,1\}^n$
- Each finite set (of states) can be mapped to $\{0,1\}^n$
- Boolean function can be presented as OBDD.
- Given Kripke structure $M = (S, S_0, R, L)$
  - Set of states S
  - Set of intitial states S_0
  - Transition relation R is set of pairs of states
  - Labelling function L, L^{-1} maps labels to sets of states
- States, intial states, transition relation and labeling can be represented as OBDDs

Symbolic CTL Model Checking

State space

- Each state can be represented as vector of boolean values
- The set of initial states can be represented by a boolean function

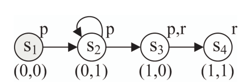


Example
- $S_0 = \{(x_1,x_2) | \neg x_1 \wedge \neg x_2\}$

Symbolic CTL Model Checking

Transition relation

- Each transition can be represented pairs of states
- The transition relation can be represented by a boolean function over $x_1,\ldots, x_n, x_1{}',\ldots, x_n{}'$



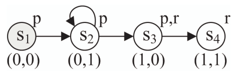
Example
- (s_1,s_2) as ((0,0),(0,1))
- ((0,0), (0,1)) satsifies $(\neg x_1 \wedge \neg x_2 \wedge \neg x'_1 \wedge x'_2)$
- R is the disjunction of boolean functions for all transitions

Symbolic CTL Model Checking

Labelling

- L^{-1} maps labels to sets of states
- This set can be represented as boolean function

Example



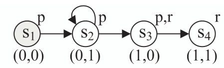
- $L^{-1}(p)=\{(0,0),(0,1),(1,0)\}$
- $L^{-1}(p)$ is represented by $(\neg x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2) = \neg x_2 \vee (\neg x_1 \wedge x_2)$

Comp4151 Ansgar Fehnker

Symbolic CTL Model Checking

Symbolic Representation

Given M = (S, S0, R, L) over set of atomic proposition AP



Explicit representation

$AP = \{p, r\}$
$S = \{s_1, s_3, s_2, s_4\}$
$S_0 = \{s_1\}$
$R = \{(s_1, s_2), (s_2, s_2), (s_2, s_3), (s_3, s_4)\}$
$L(s_1) = \{p\}$ $L(s_2) = \{p\}$
$L(s_3) = \{p,r\}$ $L(s_4) = \{p\}$

Symbolic representation

$s_1 \triangleq (0,0)$ $s_2 \triangleq (0,1)$
$s_3 \triangleq (1,0)$ $s_4 \triangleq (1,1)$
$R = \{ ((0,0), (0,1)), ((0,1), (0,1)), ((0,1), (1,0)), ((1,0), (1,1)) \}$
$L^{-1}(p) = \{(0,0), (0,1), (1,0)\}$
$L^{-1}(r) = \{(1,0), (1,1)\}$

Comp4151 Ansgar Fehnker

Symbolic CTL Model Checking

OBDD operations

Booelan operators

$[\|true\|] = S$	OBDD with leaf 1
$[\|false\|] = \emptyset$	OBDD with leaf 0
$[\| p \|] = \{ s \mid p \in \mu(s) \}$	OBDD for $L^{-1}(p)$
$[\| \neg \phi \|] = S \setminus [\| \phi \|]$	Negate(B_ϕ)
$[\| \phi \wedge \psi \|] = [\| \phi \|] \cap [\| \psi \|]$	Apply($\wedge$, B_ϕ, B_ψ)

Comp4151 Ansgar Fehnker

Symbolic CTL Model Checking

OBDD operations

Temporal Operators

- $[\| EX\, \phi \|] = \{ s \mid \exists (s,s') \in \rightarrow$ such that $s' \in [\| \phi \|] \}$
- $\exists x'_1,\ldots,x'_n (f_\phi(x'_1,\ldots,x'_n) \wedge f_R(x_1,\ldots,x_n,x'_1,\ldots,x'_n))$
- Impleme[OBDD for ϕ] combi[OBDD for transition relation R] Exists

Comp4151 Ansgar Fehnker

Symbolic CTL Model Checking

OBDD operations

Temporal Operators

- [| E ϕ U ψ |] is least fixpont of
  $f_{EU}(P)=[|\psi|] \cup ([|\phi|] \cap [|EX\ P|])$

- Implemented by combination of Apply and Exists

Symbolic CTL Model Checking

OBDD operations

Temporal Operators

- [| EG ϕ |] is the greatest fixpoint of
  $f_{EG}(P) = [|\phi|] \cap [|EX\ P|]$

- Implemented by combination of Apply and Exists

Symbolic CTL Model Checking

Summary

- Fixpoint characterization of CTL translates to an algorithm based on sets
- Set in general and Kripke structures in particular can be modelled with boolean function
- OBDDs provide a canonical representation of boolean functions that provides
  - Compact representation of sets
  - Efficient operations on sets
- Next lecture: SMV