BDD operations

Paul Jackson[1]

University of Edinburgh

Automated Reasoning
21st November 2013

[1] Diagrams from Huth & Ryan, LiCS, 2nd Ed.

reduce algorithm

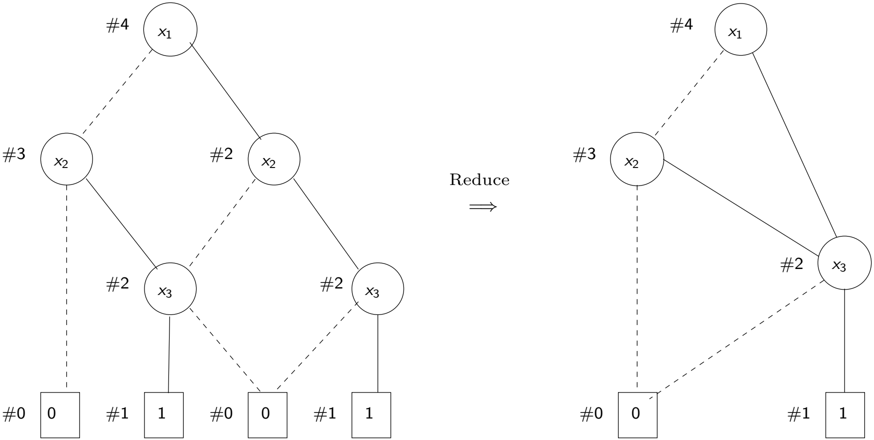
Aim is to construct a ROBDD from an OBDD.

- Adds integer labels $\mathrm{id}(n)$ to each node n of a BDD in a single bottom-up pass
- Key property:

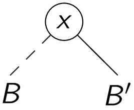
  if nodes m and n are labelled, then
  $\mathrm{id}(m) = \mathrm{id}(n)$ iff m and n represent the same Boolean function.
- Rules for adding label to node n:
  - remove duplicate terminals: if n terminal, set $\mathrm{id}(n)$ to $\mathrm{val}(n)$
  - remove redundant tests: if $\mathrm{id}(\mathrm{lo}(n)) = \mathrm{id}(\mathrm{hi}(n))$, set $\mathrm{id}(n)$ to $\mathrm{id}(\mathrm{lo}(n))$
  - remove duplicate nodes: if there exists a labelled node m such that $\left\{\begin{array}{ccc} \mathrm{var}(m) & = & \mathrm{var}(n) \\ \mathrm{id}(\mathrm{lo}(m)) & = & \mathrm{id}(\mathrm{lo}(n)) \\ \mathrm{id}(\mathrm{hi}(m)) & = & \mathrm{id}(\mathrm{hi}(n)) \end{array}\right\}$, set $\mathrm{id}(n)$ to $\mathrm{id}(m)$

    Use hash table with $\langle \mathrm{var}(n), \mathrm{id}(\mathrm{lo}(n)), \mathrm{id}(\mathrm{hi}(n)) \rangle$ keys for $O(1)$ search time
  - otherwise, set $\mathrm{id}(n)$ to unused number
- ROBDD generated by using 1 node from each class of nodes with the same label

reduce example



`apply` algorithm I

- Let op be a symbol for any binary operation on boolean formulas. (e.g. $\wedge$, $\vee$, $\oplus$)
- Given BDDs B_f and B_g for boolean formulas f and g, `apply`(op, B_f, B_g) computes a BDD for f op g.
- Can also do negation if op is $\lambda x.\, x \oplus \top$.
- If BDD 

represents a Boolean formula f,

  then sub-BDD B represents $f[0/x]$, B' represents $f[1/x]$, and have

  $$f \;\equiv\; \overline{x}.f[0/x] + x.f[1/x]$$

  This is the *Shannon expansion* of Boolean formula f with respect to the variable x
- While Sub-BDDs B and B' are drawn as distinct, in general they share structure

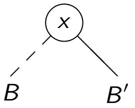
`apply` algorithm II

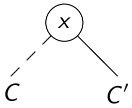
- By applying Shannon expansion to f and g in f op g and rearranging terms, we get a recursive characterisation of op.

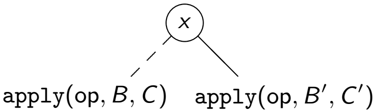
$$f \text{ op } g = \overline{x}.(f[0/x] \text{ op } g[0/x]) + x.(f[1/x] \text{ op } g[1/x])$$

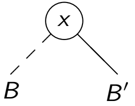
- This motivates a recursive algorithm for `apply`

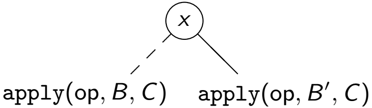
apply algorithm III

apply(op, 

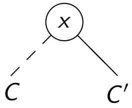
, 

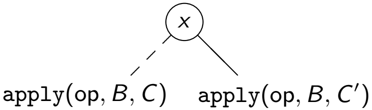
) = 

apply(op, 

, C) = 

where C is 1) a terminal node or 2) a non-terminal with $var(root(C)) > x$

apply(op, B , 

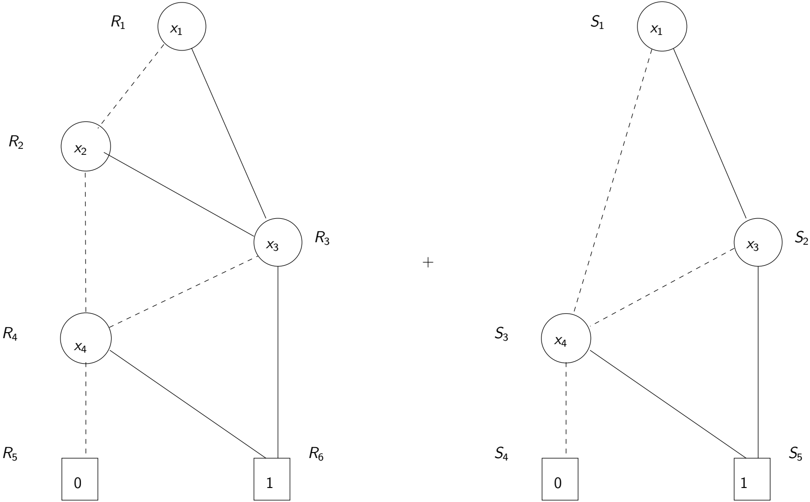
) = 

where B is 1) a terminal node or 2) a non-terminal with $var(root(B)) > x$

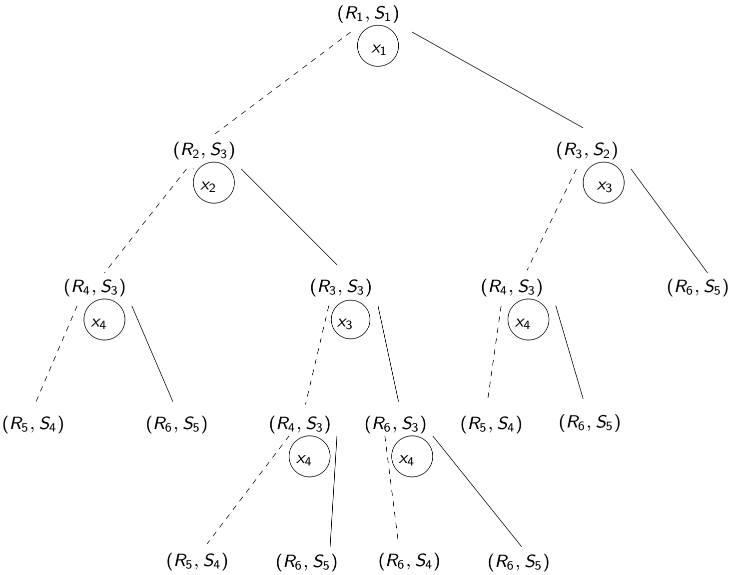
apply(op, $\boxed{u}$, $\boxed{v}$) = $\boxed{w}$ where $w = u \text{ op } v$

apply example

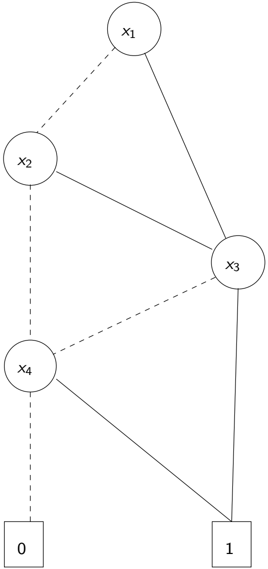
Compute $\texttt{apply}(+, B_f, B_g)$ where B_f and B_g are:



Recursive calls of `apply`



Final result from `apply` execution



`apply` remarks

- In general, result will not be an ROBDD, so need to use `reduce` afterwards
  - Or can incorporate aspects of `reduce` into `apply` so result is always reduced
- Many calls can be identical, so calls *memoized* to improve efficiency

Other operations

- `restrict`$(0, x, B_f)$ computes ROBDD for $f[0/x]$
  1. For each node n labelled with an x, incoming edges are redirected to $\mathrm{lo}(n)$ and n is removed.
  2. Resulting BDD is reduced.
- `exists`(x, B_f) computes ROBDD for $\exists x.\, f$
  - Uses identity $(\exists x.\, f) \equiv f[0/x] + f[1/x]$ and `restrict` and `apply` functions

Time complexities

Algorithm	Input OBDD(s)	Output OBDD	Time-complexity
`reduce`	B	reduced B	$O(\lvert B\rvert \cdot \log \lvert B\rvert)$
`apply`	B_f, B_g (reduced)	$B_{f \text{ op } g}$ (reduced)	$O(\lvert B_f\rvert \cdot \lvert B_g\rvert)$
`restrict`	B_f (reduced)	$B_{f[0/x]}$ or $B_{f[1/x]}$ (reduced)	$O(\lvert B_f\rvert \cdot \log \lvert B_f\rvert)$
$\exists$	B_f (reduced)	$B_{\exists x_1.\exists x_2.\dots.\exists x_n.f}$ (reduced)	NP-complete

Encoding CTL algorithms using BDDs I

- States represented using Boolean vectors $\langle v_1, \ldots, v_n \rangle$, where $v_i \in \{0, 1\}$.
- Sets of states represented using BDDs on n variables $x_1, \ldots x_n$ describing characteristic functions of sets.
- Set operations $\cup, \cap, \bar{}$ made effective using using the `apply` and the Boolean operations $+, \cdot, \bar{}$.
- Transition relations described using BDDs on $2n$ variables.
  - If Boolean variables $x_1, \ldots x_n$ describe initial state and Boolean variables $x'_1, \ldots x'_n$ describe next state, then good ordering is $x_1, x'_1, x_2, x'_2, \ldots x_n, x'_n$.
- Translations of Boolean formulas describing state sets and transition relations into BDDs make use of `apply` algorithm, following structure of formulas
  - This avoids the intractable exponential blow-up if instead one tried to first construct a binary decision tree.

Encoding CTL algorithms using BDDs II

- Function application

$$\mathrm{pre}_{\exists}(Y) \;\doteq\; \{s \in S \mid \exists s' \in S.\, s \rightarrow s' \wedge s' \in Y\}$$

is represented by the BDD

$$\texttt{exists}(\hat{x}', \texttt{apply}(\cdot, B_{\rightarrow}, B_{Y'})) \quad ,$$

where

  - $B_{\rightarrow}$ is the BDD representing the transition relation $\rightarrow$
  - $B_{Y'}$ is the BDD representing set Y with the variables $x_1, \ldots x_n$ renamed to $x'_1, \ldots x'_n$

- Function application

$$\mathrm{pre}_{\forall}(Y) \;\doteq\; \{s \in S \mid \forall s' \in S.\, s \rightarrow s' \Rightarrow s' \in Y\}$$

is represented using the identity

$$\mathrm{pre}_{\forall}(Y) \;=\; S - \mathrm{pre}_{\exists}(S - Y)$$

and the representation of $\mathrm{pre}_{\exists}(S - Y)$ and set complement.