Planning and Optimization

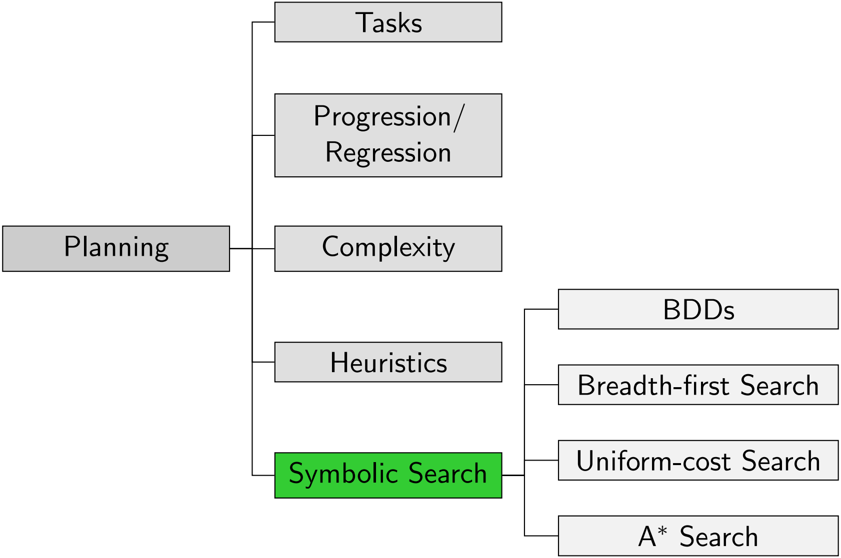
G1. Symbolic Search: BDDs

Malte Helmert and Gabriele Röger

Universität Basel

December 13, 2017

Content of this Course



Motivation

Dealing with Large State Spaces

- One way to explore very large state spaces is to use selective exploration methods (such as heuristic search) that only explore a fraction of states.
- Another method is to concisely represent large sets of states and deal with large state sets at the same time.

Breadth-first Search with Progression and State Sets

Progression Breadth-first Search

def bfs-progression(V, I, O, γ):
  $goal := formula\text{-}to\text{-}set(\gamma)$
  $reached_0 := \{I\}$
  $i := 0$
  loop:
    if $reached_i \cap goal \neq \emptyset$:
      return solution found
    $reached_{i+1} := reached_i \cup apply(reached_i, O)$
    if $reached_{i+1} = reached_i$:
      return no solution exists
    $i := i + 1$

$\rightsquigarrow$ If we can implement operations *formula-to-set*, $\{I\}$, $\cap$, $\neq \emptyset$, $\cup$, *apply* and $=$ efficiently, this is a reasonable algorithm.

Formulae to Represent State Sets

- We have previously considered boolean formulae as a means of representing sets of states.
- Compared to explicit representations of state sets, boolean formulae have very nice performance characteristics.

Note: In the following, we assume that formulae are implemented as trees, not strings, so that we can e.g. compute $\chi \land \psi$ from χ and ψ in constant time.

Performance Characteristics

Explicit Representations vs. Formulae

Let k be the number of state variables, $|S|$ the number of states in S and $\|S\|$ the size of the representation of S.

	Sorted vector	Hash table	Formula
$s \in S$?	$O(k \log \|S\|)$	$O(k)$	$O(\\|S\\|)$
$S := S \cup \{s\}$	$O(k \log \|S\| + \|S\|)$	$O(k)$	$O(k)$
$S := S \setminus \{s\}$	$O(k \log \|S\| + \|S\|)$	$O(k)$	$O(k)$
$S \cup S'$	$O(k\|S\| + k\|S'\|)$	$O(k\|S\| + k\|S'\|)$	$O(1)$
$S \cap S'$	$O(k\|S\| + k\|S'\|)$	$O(k\|S\| + k\|S'\|)$	$O(1)$
$S \setminus S'$	$O(k\|S\| + k\|S'\|)$	$O(k\|S\| + k\|S'\|)$	$O(1)$
$\overline{S}$	$O(k2^k)$	$O(k2^k)$	$O(1)$
$\{s \mid s(v) = 1\}$	$O(k2^k)$	$O(k2^k)$	$O(1)$
$S = \emptyset$?	$O(1)$	$O(1)$	co-NP-complete
$S = S'$?	$O(k\|S\|)$	$O(k\|S\|)$	co-NP-complete
$\|S\|$	$O(1)$	$O(1)$	#P-complete

Which Operations are Important?

- Explicit representations such as hash tables are not suitable because their size grows linearly with the number of represented states.
- Formulae are very efficient for some operations, but not very well suited for other important operations needed by the progression algorithm.
  - Examples: $S \neq \emptyset$?, $S = S'$?
- One of the sources of difficulty is that formulae allow many different representations for a given set.
  - For example, all unsatisfiable formulae represent $\emptyset$.

  This makes equality tests expensive.

$\rightsquigarrow$ We are interested in canonical representations, i.e. representations for which there is only one possible representation for every state set.

Binary decision diagrams (BDDs) are an example of an efficient canonical representation.

Performance Characteristics

Formulae vs. BDDs

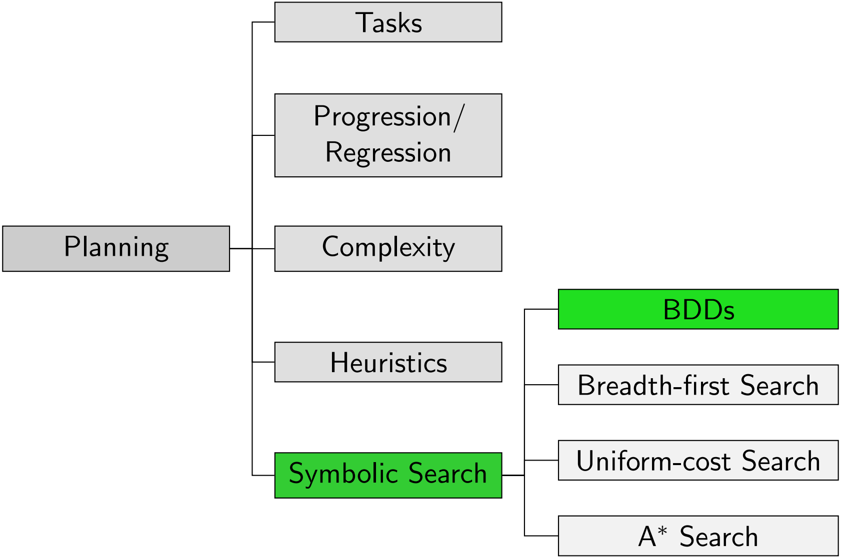
Let k be the number of state variables, $|S|$ the number of states in S and $\|S\|$ the size of the representation of S.

	Formula	BDD
$s \in S$?	$O(\|S\|)$	$O(k)$
$S := S \cup \{s\}$	$O(k)$	$O(k)$
$S := S \setminus \{s\}$	$O(k)$	$O(k)$
$S \cup S'$	$O(1)$	$O(\|S\|\|S'\|)$
$S \cap S'$	$O(1)$	$O(\|S\|\|S'\|)$
$S \setminus S'$	$O(1)$	$O(\|S\|\|S'\|)$
$\overline{S}$	$O(1)$	$O(\|S\|)$
$\{s \mid s(v) = 1\}$	$O(1)$	$O(1)$
$S = \emptyset$?	co-NP-complete	$O(1)$
$S = S'$?	co-NP-complete	$O(1)$
$\lvert S \rvert$	#P-complete	$O(\|S\|)$

Remark: Optimizations allow BDDs with complementation ($\overline{S}$) in constant time, but we will not discuss this here.

Binary Decision Diagrams

Content of this Course



Binary Decision Diagrams: Definition

Definition (BDD)

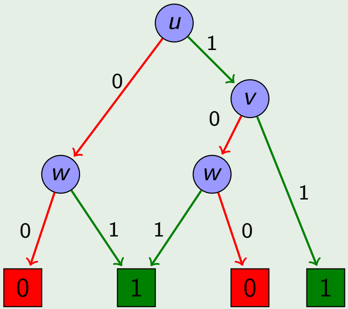
Let V be a set of propositional variables.
A binary decision diagram (BDD) over V is a directed acyclic graph with labeled arcs and labeled vertices satisfying the following conditions:

- There is exactly one node without incoming arcs.
- All sinks (nodes without outgoing arcs) are labeled 0 or 1.
- All other nodes are labeled with a variable $v \in V$ and have exactly two outgoing arcs, labeled 0 and 1.

BDD Example

Example

Possible BDD for $(u \wedge v) \vee w$



Binary Decision Diagrams: Terminology

BDD Terminology

- The node without incoming arcs is called the root.
- The labeling variable of an internal node is called the decision variable of the node.
- The nodes reached from node n via the arc labeled $i \in \{0, 1\}$ is called the i-successor of n.
- The BDDs which only consist of a single sink are called the zero BDD and one BDD, respectively.

Observation: If B is a BDD and n is a node of B, then the subgraph induced by all nodes reachable from n is also a BDD.

- This BDD is called the BDD rooted at n.

BDD Semantics

Testing whether a BDD Includes a Variable Assignment

def bdd-includes(B: BDD, a: variable assignment):
    Set n to the root of B.
    while n is not a sink:
        Set v to the decision variable of n.
        Set n to the $a(v)$-successor of n.
    return true if n is labeled 1, false if it is labeled 0.

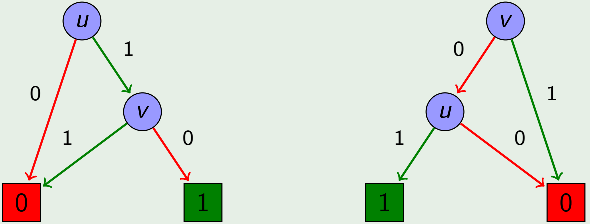
Definition (Set Represented by a BDD)

Let B be a BDD over variables V. The set represented by B, in symbols $r(B)$ consists of all variable assignments $a : V \to \{0, 1\}$ for which *bdd-includes*(B, a) returns true.

Ordered BDDs: Motivation

In general, BDDs are not a canonical representation for sets of valuations. Here is a simple counter-example ($V = \{u, v\}$):

Example (BDDs for $u \wedge \neg v$ with Different Variable Order)



Both BDDs represent the same state set, namely the singleton set $\{\{u \mapsto 1, v \mapsto 0\}\}$.

Ordered BDDs: Definition

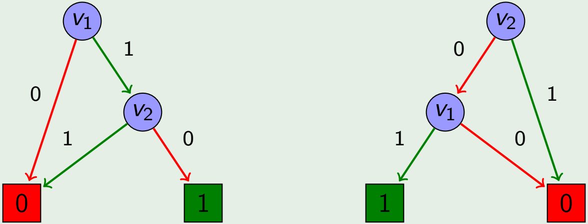
- As a first step towards a canonical representation, we will in the following assume that the set of variables A is totally ordered by some ordering $\prec$.
- In particular, we will only use variables $v_1, v_2, v_3, \ldots$ and assume the ordering $v_i \prec v_j$ iff $i < j$.

Definition (Ordered BDD)

A BDD is ordered iff for each arc from an internal node with decision variable u to an internal node with decision variable v, we have $u \prec v$.

Ordered BDDs: Example

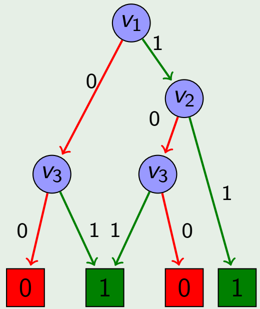
Example (Ordered and Unordered BDD)



The left BDD is ordered, the right one is not.

Reduced Ordered BDDs: Are Ordered BDDs Canonical?

Example (Two equivalent BDDs that can be reduced)



- Ordered BDDs are not canonical: Both ordered BDDs represent the same set.
- However, ordered BDDs can easily be made canonical.

Reduced Ordered BDDs: Reductions (1)

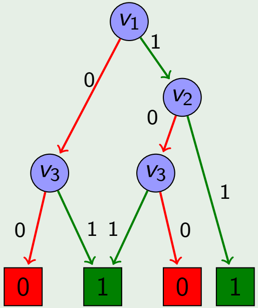
There are two important operations on BDDs that do not change the set represented by it:

Definition (Isomorphism Reduction)

If the BDDs rooted at two different nodes n and n' are isomorphic, then all incoming arcs of n' can be redirected to n, and all parts of the BDD no longer reachable from the root removed.

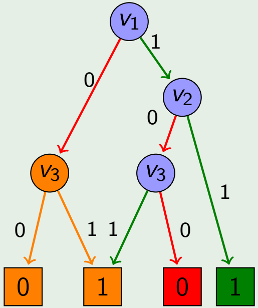
Reduced Ordered BDDs: Reductions (2)

Example (Isomorphism Reduction)



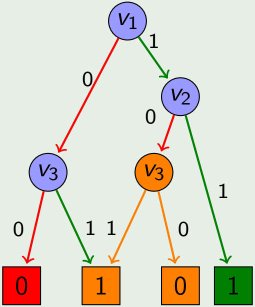
Reduced Ordered BDDs: Reductions (2)

Example (Isomorphism Reduction)



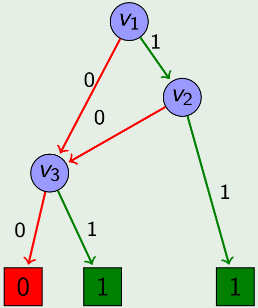
Reduced Ordered BDDs: Reductions (2)

Example (Isomorphism Reduction)



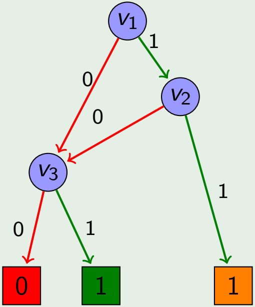
Reduced Ordered BDDs: Reductions (2)

Example (Isomorphism Reduction)



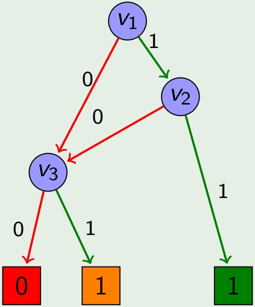
Reduced Ordered BDDs: Reductions (2)

Example (Isomorphism Reduction)



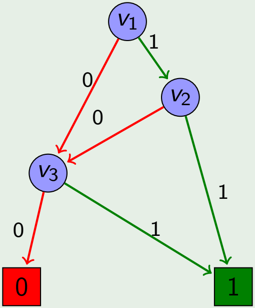
Reduced Ordered BDDs: Reductions (2)

Example (Isomorphism Reduction)



Reduced Ordered BDDs: Reductions (2)

Example (Isomorphism Reduction)



Reduced Ordered BDDs: Reductions (3)

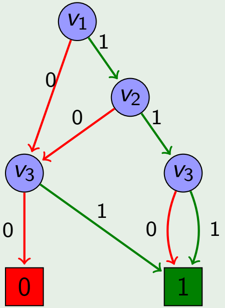
There are two important operations on BDDs that do not change the set represented by it:

Definition (Shannon Reduction)

If both outgoing arcs of an internal node n of a BDD lead to the same node m, then n can be removed from the BDD, with all incoming arcs of n going to m instead.

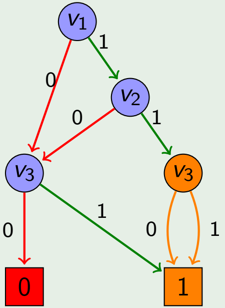
Reduced Ordered BDDs: Reductions (4)

Example (Shannon Reduction)



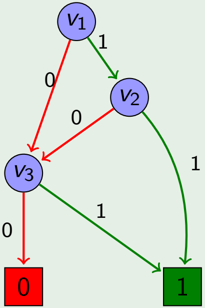
Reduced Ordered BDDs: Reductions (4)

Example (Shannon Reduction)



Reduced Ordered BDDs: Reductions (4)

Example (Shannon Reduction)



Reduced Ordered BDDs: Definition

Definition (Reduced Ordered BDD)

An ordered BDD is reduced iff it does not admit any isomorphism reduction or Shannon reduction.

Theorem (Bryant 1986)

For every state set S and a fixed variable ordering, there exists exactly one reduced ordered BDD representing S.

Moreover, given any ordered BDD B, the equivalent reduced ordered BDD can be computed in linear time in the size of B.

⇝ Reduced ordered BDDs are the canonical representation we were looking for.

From now on, we simply say BDD for reduced ordered BDD.

BDD Implementation

Efficient BDD Implementation: Ideas

- Earlier, we showed some BDD performance characteristics.
  - Example: $S = S'$? can be tested in time $O(1)$.
- The critical idea for achieving this performance is to share structure not only within a BDD, but also between different BDDs.

BDD Representation

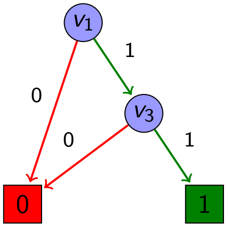
- Every BDD (including sub-BDDs) B is represented by a single natural number $id(B)$ called its ID.
  - The zero BDD has ID -2.
  - The one BDD has ID -1.
  - Other BDDs have IDs ≥ 0.
- The BDD operations must satisfy the following invariant: Two BDDs with different ID are never identical.

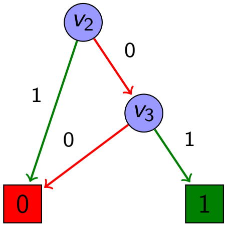
Efficient BDD Implementation: Data Structures

Data Structures

- There are three global vectors (dynamic arrays) to represent information on non-sink BDDs with ID $i \geq 0$:
  - $var[i]$ denotes the decision variable.
  - $low[i]$ denotes the ID of the 0-successor.
  - $high[i]$ denotes the ID of the 1-successor.
- There is some mechanism that keeps track of IDs that are currently unused (garbage collection, reference counting).
  - This can be implemented without amortized overhead.
- There is a global hash table *lookup* which maps, for each ID $i \geq 0$ representing a BDD in use, the triple $\langle var[i], low[i], high[i] \rangle$ to i.
  - Randomized hashing allows constant-time access in the expected case. More sophisticated methods allow deterministic constant-time access.

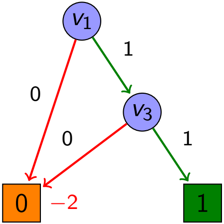
Efficient BDD Implementation: Data Structures Example



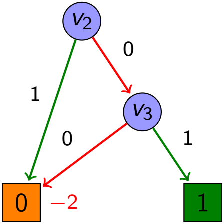


formula	ID i	$var[i]$	$low[i]$	$high[i]$

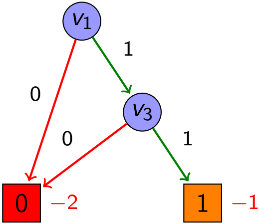
Efficient BDD Implementation: Data Structures Example

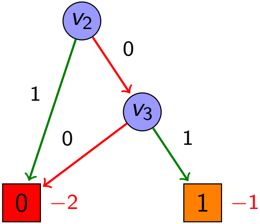


formula	ID i	$var[i]$	$low[i]$	$high[i]$
$\bot$	-2	–	–	–



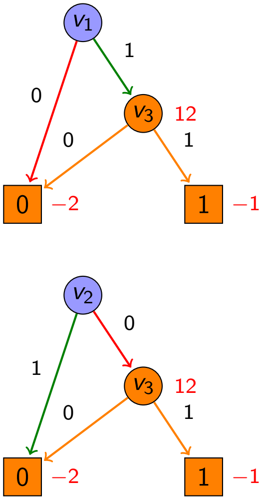
Efficient BDD Implementation: Data Structures Example





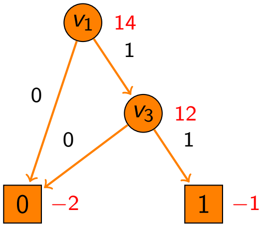
formula	ID i	$var[i]$	$low[i]$	$high[i]$
$\bot$	-2	–	–	–
$\top$	-1	–	–	–

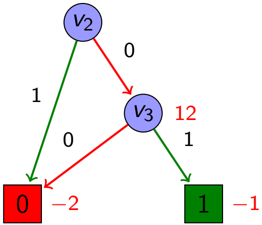
Efficient BDD Implementation: Data Structures Example



formula	ID i	$var[i]$	$low[i]$	$high[i]$
$\bot$	-2	–	–	–
$\top$	-1	–	–	–
v_3	12	3	-2	-1

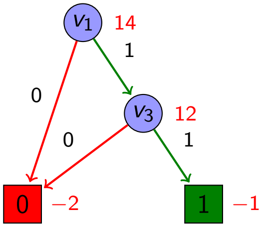
Efficient BDD Implementation: Data Structures Example

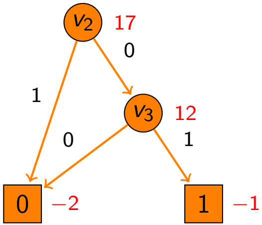




formula	ID i	$var[i]$	$low[i]$	$high[i]$
$\bot$	−2	–	–	–
$\top$	−1	–	–	–
v_3	12	3	−2	−1
$v_1 \wedge v_3$	14	1	−2	12

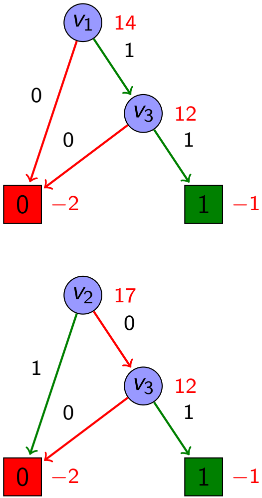
Efficient BDD Implementation: Data Structures Example





formula	ID i	$var[i]$	$low[i]$	$high[i]$
$\bot$	-2	–	–	–
$\top$	-1	–	–	–
v_3	12	3	-2	-1
$v_1 \wedge v_3$	14	1	-2	12
$\neg v_2 \wedge v_3$	17	2	12	-2

Efficient BDD Implementation: Data Structures Example



formula	ID i	$var[i]$	$low[i]$	$high[i]$
$\bot$	−2	–	–	–
$\top$	−1	–	–	–
v_3	12	3	−2	−1
$v_1 \wedge v_3$	14	1	−2	12
$\neg v_2 \wedge v_3$	17	2	12	−2

Building BDDs (1)

Building the Zero BDD

```
def zero():
    return −2
```

Building the One BDD

```
def one():
    return −1
```

Building BDDs (2)

Building Other BDDs

def bdd(v: variable, l: ID, h: ID):
  if $l = h$:
    return l
  if $\langle v, l, h \rangle \notin lookup$:
    Set i to a new unused ID.
    $var[i], low[i], high[i] := v, l, h$
    $lookup[\langle v, l, h \rangle] := i$
  return $lookup[\langle v, l, h \rangle]$

We only create BDDs with zero, one and bdd (i.e., function bdd is the only function writing to *var*, *low*, *high* and *lookup*). Thus:

- BDDs are guaranteed to be reduced.
- BDDs with different IDs always represent different sets.

BDD Operations

This representation allows to implement all operations so that the following performance characteristics are met.

	BDD
$s \in S?$	$O(k)$
$S := S \cup \{s\}$	$O(k)$
$S := S \setminus \{s\}$	$O(k)$
$S \cup S'$	$O(\|S\|\|S'\|)$
$S \cap S'$	$O(\|S\|\|S'\|)$
$S \setminus S'$	$O(\|S\|\|S'\|)$
$\overline{S}$	$O(\|S\|)$
$\{s \mid s(v) = 1\}$	$O(1)$
$S = \emptyset?$	$O(1)$
$S = S'?$	$O(1)$
$\lvert S \rvert$	$O(\|S\|)$

Implementation details in next chapter.

Summary

Summary

- Binary decision diagrams are a data structure to compactly represent and manipulate sets of variable assignments.
- Reduced ordered BDDs are a canonical representation of such sets.
- An efficient implementation shares structure between BDDs.