Using SAT-solver for FBDD-construction

Robert Wille
Outline

- Introduction
  - What are DDs?
  - Why DDs?
  - Main problem with DDs
  - What is common?

- Motivation

- How can a SAT-solver help?

- Challenges

- Conclusion
What is a decision diagram?

- data structure that is used to represent a Boolean function
- rooted, directed, acyclic graph, which consists of decision nodes and two terminal nodes (leaves)
- each decision node is labeled by a Boolean variable and has two child nodes (low and high)
DD - Example

\[ f = x_1 x_2 + x_3 \]

- to calculate the children nodes
  \[ f = \overline{x}_i \ f_{x_i=0} + x_i \ f_{x_i=1} \]
  (Shannon decomposition)
Why DDs? #1

- Efficient representation of Boolean functions
  - Canonical (under constraints, more later)
  - Compact
  - Manipulable

- Used in countless applications, i.e.
  - Synthesis (generation of 100% testable circuits)
  - Supporting DPLL
  - ATPG
  - ...
Why DDs? #2

- used in countless applications, i.e.
  - generation of 100% testable circuits
ordered binary decision diagram (OBDD)
- on all paths the variables occur in the same order
- variable ordering influences the size of DD

f = x₁x₃ + x₂x₄
main problem

memory explosion

How we can make DDs smaller?
reduction rules of BDDs #1

- merge isomorphic subtrees
reduction rules of BDDs #2

- delete nodes, if they have the same successors at both children
reduction of DDs – example #1
reduction of DDs – example #2
reduction of DDs – example #3
reduction of DDs – example #4
reduction of DDs – example #4
BDDs are canonical

- reduced ordered binary decision diagramms are also know as binary decision diagram (BDD) for short

- Bryant, 86: two Boolean functions are equivalent, if there BDDs are isomorphc for a given variable order

- state-of-the-art: CUDD-Package
Outline

- Introduction
  - What are DDs?
  - Why DDs?
  - Main problem with DDs
  - What is common?

- Motivation

- How can a SAT-solver help?

- Challenges

- Conclusion
motivation

- remember: How we can make DDs smaller? (memory explosion)

- BDDs:
  - reduction rules
  - finding good order (NP-hard)

- using DDs with fewer restrictions
  - free binary decision diagrams (FBDD)
FBDDs

- may have different orderings along each path
  → could lead to smaller DDs

- problem now: finding good order for each path
What is a SAT-solver? #1

- an algorithm for solving Boolean Functions

- gets a Boolean Function over $n$ variables and returns
  - UNSAT, if the function is unsatisfiable
  - SAT, otherwise
    - in the case of SAT, the solver will provide one satisfying assignment as well
What is a SAT-solver? #2

- Boolean Function will provided as conjunctive normal form
  - finite set of variables $S$
  - literal: variable $x$ or its negation
  - clause: disjunction of literals
  - CNF: conjunction of clauses

$$(x_1 + x_2)$$
$$(x_3 + x_4)$$
$$(x_1 + x_3 + \overline{x_4})$$
What is a SAT-solver? #3

- an algorithm for solving Boolean functions

```plaintext
do
    propagate();
    if (conflict)
        analyzeAndBT();
    else
        decide();
while (non_solution_found)
```
How can a SAT-solver help? #1

do
  propagate();
  if (conflict)
    analyzeAndBT();
  else
    decide();
while (non_solution_found)
How can a SAT-solver help? #2

- search space exploration of SAT-solver could provide a good FBDD

- why maybe better than other approaches?
  - SAT is a well known problem
  - highly optimized algorithms and heuristics exist
Implementation

- using state-of-the-art SAT-solver and implement FBDD-construction on the top of it

- in a primitive version
  - instruct the algorithm to find all solution
  - construct FBDD while searching
Outline

- Introduction
  - What are DDs?
  - Why DDs?
  - Main problem with DDs
  - What is common?

- Motivation

- How can a SAT-solver help?

- Implementation

- Challenges

- Conclusion
finding isomorphic subtrees

...is quite harder for FBDDs than for BDDs
special conflict clauses

- using 'special' conflict clauses

- pretend a conflict here

- creating 'special' conflict clause which is caching subtree

- if these becomes conflicted use cached subtree

search space completely explored
discussion: special conflict clses

- very cheap in general

- not sure, that we will get all isomorphic subtrees
  → but maybe the most?
using cutlines #1

using cutlines #2
discussion: using cutlines

- restriction that only inputs could make decisions

- not possible to use output as unit clause

→ only propagations from inputs are possible
→ this may increase the search space/FBDD-size
using cutsets

\[ c_5: v_5 + v_6 \]
\[ c_4: v_4 + \neg v_5 + v_6 \]
\[ c_3: v_1 + v_3 + v_4 + v_5 \]
\[ c_2: v_2 + v_3 \]
\[ c_1: v_1 + v_2 + \neg v_3 \]

Huang, Jinbo; Darwiche, Adnan (2004): Using DPLL for Efficient OBDD Construction
Outline

- Introduction
  - What are DDs?
  - Why DDs?
  - Main problem with DDs
  - What is common?

- Motivation

- How could a SAT-solver help?

- Implementation

- Challenges

- Conclusion
further possible approaches

- finding special heuristics optimized for FBDD construction
- using more learning (maybe incremental SAT) to optimize an existing solution
- ...
conclusion

- often FBDDs can be better than BDDs
- the best BDD is just as good as the best FBDD (because a BDD is a special FBDD)
- maybe with the help of SAT we are able to find smaller Shannon-DDs in general
questions?

- Introduction
  - What are DDs?
  - Why DDs?
  - Main problem with DDs
  - What is common?

- Motivation

- How can a SAT-solver help?

- Implementation

- Challenges

- Conclusion