Decision Diagrams for Linear Arithmetic

Arie Gurfinkel, SEI
Sagar Chaki, SEI
Ofer Strichman, Technion/SEI
• NO WARRANTY

• THIS MATERIAL OF CARNEGIE MELLON UNIVERSITY AND ITS SOFTWARE ENGINEERING INSTITUTE IS FURNISHED ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

• Use of any trademarks in this presentation is not intended in any way to infringe on the rights of the trademark holder.

• This Presentation may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

• This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.
Linear Decision Diagrams (LDDs)

... are Binary Decision Diagrams with nodes labeled by linear inequalities

Our contributions:

- implementation on top of CUDD, including
  - support for propositional operations (AND, OR, NOT, ITE)
  - support for projection (i.e., existential quantification, QELIM) of numeric variables
  - dynamic variable ordering (DVO)
- benchmark and experiments
Motivation(1): Predicate + Numeric Abstractions

Predicate and Numeric Abstractions

Predicate Abstraction (PA) (e.g., SDV)
- Typical property: no lock is acquired twice
- Program verification reduced to propositional reasoning with model checker
- Works well for control-driven programs
- Works poorly for data-driven programs

Numeric Abstraction (NA) (e.g., ASTREE)
- Typical property: no arithmetic overflow
- Program verification reduced to arithmetic reasoning
- Works well for data-driven programs
- Works poorly for control-driven programs

How to combine PA and NA to get the best of both?
Motivation (2): Numeric Decision Diagrams

Numeric Decision Diagrams

NDD elements are

BDDs over Predicate and Numeric Terms

\[(p_1 \&\& p_2) \| (x<0 \&\& y=z)\]

\[b_1: x\geq 0, b_2: z>0\]

1-edges are black, 0-edges are red
edges to 0 node are not shown

\[b_1:x\geq 0, b_2:z>0, b_3:y=z\]

Motivation (2): Numeric Decision Diagrams

Problems with NDDs are:
• No reduction w.r.t. the types of constraints
• All numeric operations are done path-at-a-time (i.e., exponential in the diagram!!!)

Lesson learned: need diagrams for linear arithmetic with efficient (not path-at-a-time) existential quantification
Some Other Applications of LDDs

• Represent and manipulate Boolean formulas over linear arithmetic ...
  – to compute predicate abstraction
  – to summarize loop-free code
  – for program analysis with disjunctive abstract domain
  – to combine predicate and numeric abstractions
  – for timed automata verification
  – ...

• LDDs are NOT good for SATISFIABILITY checking
  – not a substitute for an SMT solver
Talk Outline

• The basics
  – variable ordering, reduction rules, propositional operations

• Dynamic Variable Ordering

• Quantifier Elimination
  – existential quantification of a single variable
  – heuristics for quantifying multiple variables

• Implementation, Benchmarks, Results

• Conclusion and Future Work
Canonizing Linear Inequalities

Linear Inequality: \( a_1 x_1 + ... + a_n x_n \leq k \)

- \( x < 10 \) \( \equiv \) \( x \leq 9 \)
- \( x + y = 10 \) \( \equiv \) \( x + y \leq 10 \land x + y \geq 10 \)
- \( x + y \geq 10 \) \( \equiv \) \( -x - y \leq -10 \)
- \( -x - y \leq -10 \) \( \equiv \) \( \neg (x + y \leq 9) \)
LDD Node Ordering

Random:
\{x \leq 0\} \{x \leq 10\} \{y \leq 5\} \{x \leq 5\} \{z \leq 6\} \{y \leq 3\}

Term-sorted:
\{x \leq 0\} \{x \leq 10\} \{x \leq 5\} \{y \leq 5\} \{y \leq 3\} \{z \leq 6\}

Ordered:
\{x \leq 0\} \{x \leq 5\} \{x \leq 10\} \{y \leq 3\} \{y \leq 5\} \{z \leq 6\}

Ordered:
\{y \leq 3\} \{y \leq 5\} \{z \leq 6\} \{x \leq 0\} \{x \leq 5\} \{x \leq 10\}
Reduction: Different Children

$t \leq 5$

Reduces to

Same as BDD
Reduction: Imply High

\[ t \leq 5 \rightarrow t \leq 9 \]

Reduces to

\[ t \leq 5 \]
Reduction: Imply Low

\[ t \leq 5 \rightarrow t \leq 9 \]
Propositional Operations: APPLY

- **ROLDD**
- **t \leq 5**
  - commutative binary operator that distributes over ITE
  - **OP**
- **t \leq 9**
  - **ROLDD**
Propositional Operations: APPLY

\[ t \leq 5 \]

\[ t \leq 9 \]
Rudell’s DVO Algorithm for BDDs

Unique Table

<table>
<thead>
<tr>
<th>Level</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$a$</td>
</tr>
<tr>
<td>1</td>
<td>$b$ $b$</td>
</tr>
<tr>
<td>2</td>
<td>$c$ $c$</td>
</tr>
<tr>
<td>3</td>
<td>$d$ $d$</td>
</tr>
</tbody>
</table>

*Edges to 0 and 1 are not shown*
Example: Changing levels

(u, (v, f11, f10), (v, f01, f00)) is overwritten in place by (v, (u, f11, f01), (u, f10, f00))

Trivial new cofactors are reduced, i.e., when f00=f10 or f01=f11

Only the diagram rooted at u is changed (both the label and the children are new)

**Complexity:** linear in the number of nodes labeled with u in the unique table
Example: Changing levels in ROLDD

ROLDD with order: u, t<=5, t<=10
(shown as a tree)

New order: t<=5, t<=10, u
Not reduced!

Cannot use BDD reordering for LDD!
Problems extending DVO to LDDs

• Broken ordering constraints
  – Solution: swap adjacent terms instead of adjacent levels
• Broken Imply-high and Imply-low rules
  – Solution: enforce the rules in swapInPlace
• LDDs are not canonical
  – Solution: LDDs are “canonical enough”: in SwapInPlace the root node is never reduced
Pairwise Swaps
Two techniques for QELIM

• **Black Box:** use QELIM for conjunctions as a black box. Apply it to all paths of a diagram
  – linear in the number of paths == exponential in the size of the diagram!
  – many examples in the literature. (e.g., we used it in NDDs)

• **White Box:** Extend Fourier-Motzkin QELIM to the DAG of LDD
  – in the best case, same complexity as BDD quantification
Fourier-Motzkin QELIM

**FM1** (Var y, Conjunction \( \varphi \))

let S be all constraints with y
remove S from \( \varphi \)
add all pairwise resolutions of S to \( \varphi \)

\[
\exists\ y . \ x-y \leq 5 \land x-z \geq 8 \land y-z \leq 10
\]

\[
x-z \geq 8 \land x-z \leq 15
\]

**FM2** (Var y, Formula \( \varphi \))

while exists constraint c with y in \( \varphi \) do
    remove c from \( \varphi \)
    resolve c with remaining constraints in \( \varphi \)
end while
WB_EXISTS1: Example

\[ \exists y \cdot x - y \leq 9 \]
WB.Exists1: Example (Cont)

\[ x - y > 9, \]

\[ x - y \leq 9, \]

\[ \exists y \cdot \]

DAG_RESOLVE (Constraint \( c \), ROLDD \( f \)) : ROLDD
Recursively resolves \( c \) with all constraints in \( f \)
Each node in \( f \) is visited only once
Quantifying out multiple variables

1: EXISTS(LDD f, Vars V)
2:   res = f;
3:   while (V != empty)
4:     V' = FIND_DROP_VARS(V, res);
5:     if (V' != empty)
6:       res = DC(CONS_OF(V'), res);
7:     V = V \ V';
8:     continue;
9:   u = PICK_VAR(V, res);
10:  res = WB_EXISTS1(u, res);
11:  V = V \ {u};
12: end while
13: return res;

EXISTS1 -- any quantification procedure that can eliminate a single variable. In our implementation, it is the optimized WB_EXISTS1 from previous slides

DC short for DROP_CONS

FIND_DROP_VARS(V, res) – finds all variables in V that have trivial resolutions on all 1-paths of res

PICK_VAR(V, res) -- picks a variable from V to be quantified out next

In our implementation, FIND_DROP_VARS and PICK_VAR are based on looking at the set of all constraints that are in support of res.
The Implementation

LDD Engine

CUDD

adapted to support DVO with LDDs

Linear Arith Theories

TVPI(Q), UTVPI(Q), UTVPI(Z)
Benchmark: Image Computation

Test case:  $\exists \, V \cdot R(V, V')$

• Each test case is constructed
  – from open source software: CUDD, mplayer, bzip2,...
  – extracted using LLVM into SSA with optimizations, aggressive loop-unrolling, and inlining
  – approximated using UTVPI constraints
• Stats: 850 test cases
  4KB – 700KB (in SMT-LIB format),
  30 – 7,956 variables

transition relation of a loop-free program fragment
With DVO

BDD = LDD: 114  BDD > LDD: 596
LDD > BDD: 140  BDD > 10*LDD: 39
Timeouts: LDD=5  BDD=28

With SVO

BDD = LDD: 407  BDD > LDD: 190
LDD > BDD: 253  LDD > 10*BDD: 18
Memory Outs: LDD=99  BDD=97
Timeouts: LDD=13  BDD=12
# Overall Results for QELIM

<table>
<thead>
<tr>
<th>Alg.</th>
<th>Hard (154 cases)</th>
<th>Easy (696 cases)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total (sec)</td>
<td>QE (sec)</td>
</tr>
<tr>
<td>BB</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>WB+SVO</td>
<td>38,739</td>
<td>36,511</td>
</tr>
<tr>
<td>WB+DVO</td>
<td>10,953</td>
<td>3,329</td>
</tr>
</tbody>
</table>
Total Time: WB+DVO and WB+SVO

<table>
<thead>
<tr>
<th>Condition</th>
<th>Easy</th>
<th>Hard</th>
</tr>
</thead>
<tbody>
<tr>
<td>WB+DVO &gt; WB+SVO</td>
<td>27</td>
<td>1</td>
</tr>
<tr>
<td>WB+DVO = WB+SVO</td>
<td>659</td>
<td>9</td>
</tr>
<tr>
<td>WB+DVO &lt; WB+SVO</td>
<td>10</td>
<td>144</td>
</tr>
</tbody>
</table>

Normalized Total Time Sorted by WB+SVO

Test Cases

Easy Cases

Hard Cases
Predicate Abstraction with LDDs

\[ \exists V \cdot R(V) \land \land_i (b_i \leftrightarrow p_i(V)) \]

- transition relation of a loop-free program fragment
- Boolean variable
- predicate
Predicate Abstraction with LDDs

\[ \exists V \cdot R(V) \land \bigwedge_i (b_i \leftrightarrow p_i(V)) \]

Running Time: MSAT and LDD
Related Work

Decision Diagrams (over linear constraints)
- Strehl. *Interval Diagram Techniques...* 1999
- Larsen et al. *Clock Difference Diagrams*. 1999

Quantifier Elimination in Large Boolean Formulas
- Clarke et al. *SATABS: A SAT-Based PA for ANSI-C*. 2005
- Cavada et al. *Computing PA by Integrating BDDs and SMT Solver*. 2007
Future Work

• Predicate Abstractions with LDDs

• An LDD-based Abstract Domain
  – first step is a disjoint-box domain for variable range analysis
  – designing a widening is the main challenge

• Public release of the library
  – send email to arie@cmu.edu for more info
THE END