Simplification of Boolean Affine Formulas

Paul Feautrier

ENS de Lyon
Paul.Feautrier@ens-lyon.fr

December 6, 2011

Boolean Affine Formulas

- Affine Atoms

$$\sum_{i=1}^{n} a_i x_i + a_0 \geq 0, \quad \sum_{i=1}^{n} a_i x_i + a_0 > 0.$$

  a_i integers, x_i integers or rationals.

- Boolean Affine Formulas: affine and boolean atoms combined by the usual connectives:
  - $\wedge$ and,
  - $\vee$ or,
  - $\neg$ not,
  - $\Rightarrow$ implies,
  - $\equiv$ equivalence
  - **true**, **false**

The If-Then-Else Operator

An interesting ternary connective:

$$\textbf{if } x \textbf{ then } y \textbf{ else } z \equiv \textbf{ite}(x, y, z).$$

ite with **true** and **false** are complete for Boolean algebra:

$$\begin{aligned} \neg x &= \textbf{ite}(x, \textbf{false}, \textbf{true}), \\ x \wedge y &= \textbf{ite}(x, y, \textbf{false}), \\ x \vee y &= \textbf{ite}(x, \textbf{true}, y). \end{aligned}$$

Note Shannon formula:

$$f(a_1, \ldots, a_n) = \textbf{ite}(a_1, f(\textbf{true}, a_2, \ldots, a_n), f(\textbf{false}, a_2, \ldots, a_n)).$$

Simplification

Eliminate useless atoms: a_1 can be eliminated if there exists a boolean function g such that:

$$f(a_1, \vec{a}) \equiv g(\vec{a}).$$

- Important for hardware synthesis: evaluating an affine atom is costly.
- Elimination is possible because affine atoms are not necessarily independent.

Complexity

Complete simplification is at least as difficult as satisfiability testing: an unsatisfiable formula must simplify to **false**.
Two approaches:

- Incomplete simplification, may have better complexity. No known algorithm yet. Dillig et. al. (SAS 2000) is still exponential.
- Take advantage of the peculiarities of the subject formula to be less than exponential in practice.

Binary Decision Diagrams

A formula using only the **ite** coonective is a Binary Decision Diagram, and look like a tree, with **true** or **false** leaves.

- A BDD is ordered if atoms always occur in the same order on all paths.
- A BDD is reduced if one has applied the reduction rule:

$$\textbf{ite}(x, y, y) \to y.$$

Properties of ROBDD

- The concept of ROBDD is a mathematical structure, which can accomodate many kinds of atoms: boolean variables, affine formulas, polynomials, etc.
- Some properties depend on the nature of the atoms, but most are universal
- Pure boolean OBDD are a canonical form for Boolean formulas
- This no longer true for affine OBDD
- There are many methods for constructing ROBDD (c.f. Shannon formula)
- ROBDD may have exponential size, but are much smaller in practice.

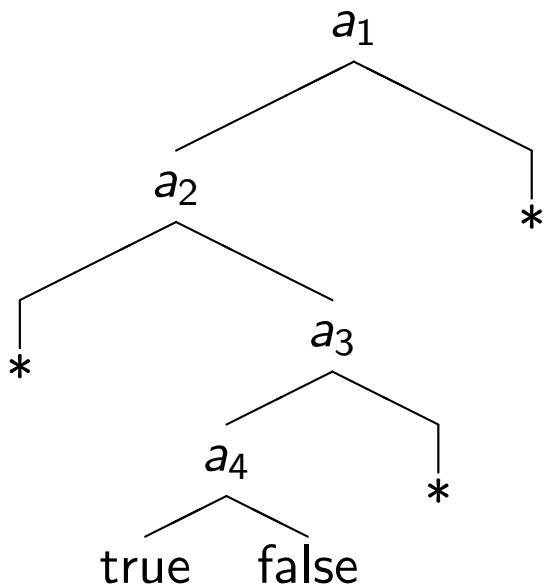
Order

The size of an ROBDD strongly depends on the atoms order.

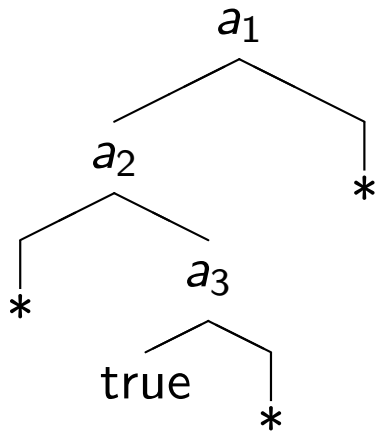
- There are examples which have exponential size – or not – depending on the order
- There are also exceptional cases which have exponential size whatever the order (integer multiplication, population counting)
- There are algoritms and heuristics for selecting the best possible order.

Simplification of Boolean-Affine OBDD

Simplification is done by path cutting.

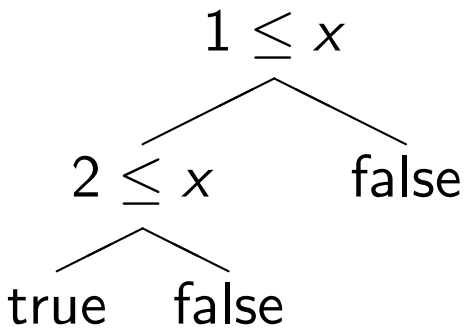


Assume that $a_1 \wedge \overline{a_2} \wedge a_3 \wedge \overline{a_3}$ is unfeasible. The right leaf under a_4 will never be reached, and can be removed with the test.

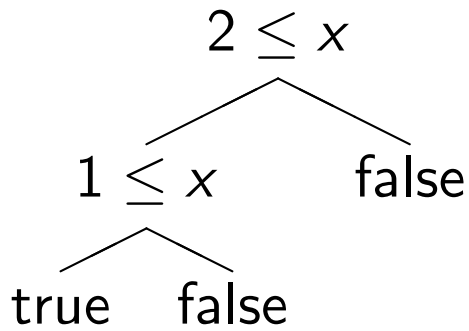


Order and Simplification

Whether a path can be cut may depend on the atoms order:



Does not simplify.



Simplifies to $2 \leq x$.

However, one does not have to check all $n!$ orders. It is enough to have each atom occupy the last position in turn.

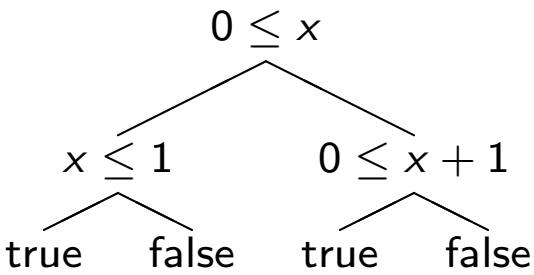
Algorithm

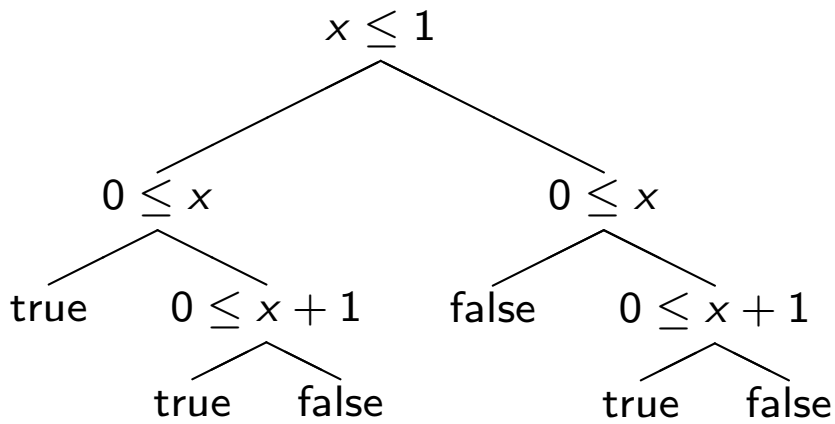
Build the formula ROBDD using an arbitrary order

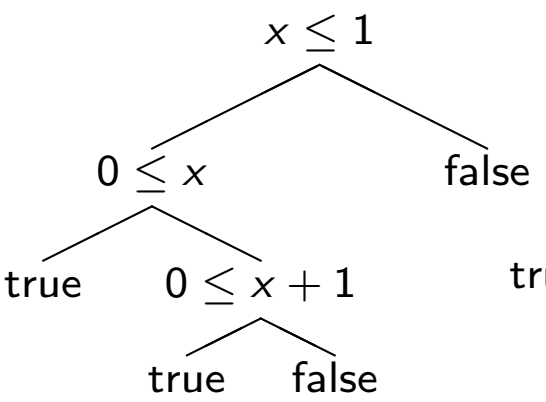
- Repeat n times
  - Bring the last atom to the top
  - For each path
    - If the path is unfeasible, cut its last step and remove its last test

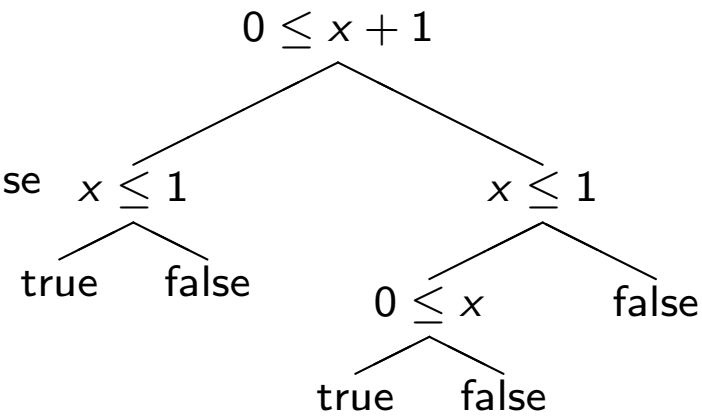
Extract the simplified formula from the last ROBDD

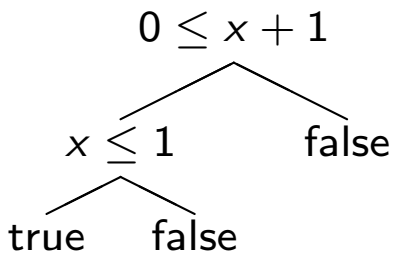
An Example











$$\textbf{ite}(x \geq 0, x \leq 1, x \geq -1) \equiv -1 \leq x \leq 1$$

Implementation

- The method has been implemented as a Java library, and is available from the author on demand.
- Feasibility testing uses a home-made SMT solver, built over the Piplib linear programming tool, based on Smullyan's semantic tableaux.
- The software has been tested on small examples from the litterature, on "quasts" generated by C. Alias' synthesis tool, with good results, and on random formulas courtesy of David Monniaux, with less impressive results.

Conclusion and Future Work

As expected, the method takes advantage of the peculiarities of the subject formula, but works poorly on random formulas.
It can be used as a post processing step for algorithms which generate Boolean affine formulas, like static program analysis, symbolic model checking, hardware and code generation.
Future work:

- Add a pure Boolean simplification step (redundant for hardware synthesis)
- Try to minimize the size of the initial BDD
- Atom coalescing: $x = 0 \vee x > 0 \equiv x \geq 0$.
- Improve the implementation (use an off-the-shelf SAT solver, avoid multiple internal representations).