Relevancy Propagation

Leonardo de Moura and Nikolaj Bjørner

Microsoft Research, One Microsoft Way, Redmond, WA, 98074, USA
{leonardo, nbjorner}@microsoft.com

Abstract. SMT solvers that perform search over a large set of constraints need to maintain, update and propagate truth assignments to atomic constraints. Each new truth assignment may lead to additional constraint propagation, which depending on the constraint domain can be costly. Relevancy propagation keeps track of which truth assignments are essential for determining satisfiability of a formula. Atoms that are marked as *relevant* have their truth assignment propagated to theory solvers, but we can avoid propagating truth assignments for atoms that are not marked as relevant.

1 Introduction

SMT solvers that perform search over a large set of constraints need to maintain, update and propagate truth assignments to atomic constraints. Each new truth assignment may lead to additional constraint propagation, which depending on the constraint domain can be costly. Examples of such domains are real or integer linear arithmetic, bit-vectors, quantifier instantiation. It is thus, for those expensive constraint domains, very desirable to limit constraint propagation to only cases that are relevant for solving the constraints.

We here explain a notion of relevancy in the context of an efficient SMT solver framework. The relevancy propagation used in Z3 was first mentioned in [1] where it provides for a drastic reduction in the search space covered during quantifier instantiation. We also found that relevancy propagation has a profound effect when reasoning about bit-vectors.

Consider the following formula:

$$a < 1 \ \vee \ (a + b > 0 \ \wedge \ b < 0)$$

It is a disjunction that requires either a to be less than 1, or to requires $a + b$ to be strictly greater than 0, but b to be less than 0. Assume that a and b range over integers, so that the legal values for a and b are the numbers $\ldots, -2, -1, 0, 1, 2, \ldots$. The formula is satisfiable. A satisfying assignment is $a \mapsto 0, b \mapsto 3$. The assignment satisfies the first disjunction, but it does not satisfy the second disjunction. A satisfying assignment for the second disjunction is $a \mapsto 2, b \mapsto -1$. The truth value of the atom $a + b > 0$ is irrelevant when satisfying the first disjunction, and it is a potential waste of resources to satisfy either $a + b > 0$ or the negation $a + b \leq 0$.

Traditional approaches to combining theory solvers with efficient SAT solvers do not have mechanisms for avoiding the un-necessary propagation of irrelevant atoms. They indiscriminatly propagate theory constraints based on truth assignments chosen by the SAT solver.

2 Proof search calculi

To motivate the technique of relevancy propagation we will here survey the essence of two popular proof-search calculi. The first calculus, called the Tableau calculus, creates a proof-search tree by decomposing an input formula into pieces. The second calculus, the DPLL calculus creates a proof-search tree by case splitting on truth values of the propositional atoms in a formula. It disregards the formula structure. Both calculi are presented as refutation calculi. By this we mean that in order to prove that an assertion φ is valid we create the negation, $\neg\varphi$, and try to derive a contradiction, or find a model for $\neg\varphi$.

2.1 Tableau search

Tableau proof search engines retain some of the structure of the input formula as an and-or tree. A tableau style search proceeds by cases: to refute a disjunction, each disjunct is refuted independently. Refuting a conjunction only requires retaining each conjunct. Conjunctions can be represented by negated disjunctions by using the de-Morgan rules. A branch is contradictory if it contains both a formula and it's negation. Tableau rules for the main propositional connectives can be summarized below:

$$\frac{\bigvee_{i=1}^{k} \varphi_i}{\varphi_1 \mid \cdots \mid \varphi_k}\ \vee \qquad \frac{\neg \bigvee_{i=1}^{k} \varphi_i}{\neg\varphi_1, \ldots, \neg\varphi_k}\ \neg\vee$$

$$\frac{\neg\neg\varphi}{\varphi}\ \neg\neg$$

$$\frac{\varphi \leftrightarrow \psi}{\varphi, \psi \mid \neg\varphi, \neg\psi}\ \leftrightarrow \qquad \frac{\neg(\varphi \leftrightarrow \psi)}{\varphi, \neg\psi \mid \neg\varphi, \psi}\ \neg\leftrightarrow$$

$$\frac{ite(\varphi_1, \varphi_2, \varphi_3)}{\varphi_1, \varphi_2 \mid \neg\varphi_1, \varphi_3}\ ite \qquad \frac{\neg ite(\varphi_1, \varphi_2, \varphi_3)}{\varphi_1, \neg\varphi_2 \mid \neg\varphi_1, \neg\varphi_3}\ \neg ite$$

- The $\vee$-rule takes a disjunction of formulas $\varphi_1, \ldots, \varphi_k$ and creates k branches. In order for the disjunction to be unsatisfiable each disjunct must be contradictory, hence the k branches.
- The $\neg\vee$-rule takes a negated disjunction and creates k new formulas in the same branch. The negated disjunction is contradictory if some combination of the constituents is contradictory.

– The $\neg\neg$-rule removes a double negation.
– The rules for bi-implication create two branches. In the positive case, the branches cover the conditions where both φ and ψ hold, or both φ and ψ don't hold. In the negative case the branches cover the conditions where φ holds, but ψ does not, or vice versa.
– The rules for if-then-else (called ite) are motivated in a similar way as the other rules.

The tableau search has the side-effect of eliminating irrelevant formulas from the scope of a branch. For example, to derive a contradiction for a disjunction $\bigvee_i \varphi_i$ the search examines each disjunction. No information is propagated or required about other disjuncts.

2.2 Davis Putnam Longman Loveland search (DPLL)

DPLL search proceeds by case splits on atomic sub-formulas appearing in the goal $\neg\varphi$. A simplistic way to characterize DPLL is by the *decide* rule:

$$\frac{\neg\varphi[p]}{\neg\varphi[\mathit{true}] \quad | \quad \neg\varphi[\mathit{false}]}\ \text{decide}$$

To refute $\neg\varphi$, which contains the propositional atom p reduce $\neg\varphi[p]$ by replacing p by *true* and by replacing p by *false*. If both reduced formulas are contradictory, then the original formula is contradictory.

Efficient implementations of DPLL operate on formulas in conjunctive normal form (CNF). CNF formulas consist of a set of clauses, each clause represents a disjunction of literals.

DPLL can be extended to handle non-propositional problems by accumulating the truth assignments to atomic formulas and make these available to theory solvers that understand only how to handle truth assignments to atoms. These extensions are commonly referred to as DPLL(T) [2].

3 Relevancy propagation

We saw how DPLL(T)-based solvers do not have the isolation property enjoyed by Tableau proof systems, as the search assigns a Boolean value to potentially all atoms appearing in a goal. For example, when clausifying $\ell_1 \vee (\ell_2 \wedge \ell_3)$ using a Tseitin [3] style algorithm we obtain the set of clauses (the last clause can be omitted while preserving satisfiability):

$$\{\ell_1, \ell_{aux}\}, \{\ell_2, \neg\ell_{aux}\}, \{\ell_3, \neg\ell_{aux}\}, \{\ell_{aux}, \neg\ell_2, \neg\ell_3\}\ .$$

Now, suppose that ℓ_1 is assigned true. In this case, ℓ_2 and ℓ_3 are clearly irrelevant and truth assignments to ℓ_2 and ℓ_3 need not be propagated to the theory solvers, but the Tseitin encoding, which creates a set of clauses, makes the act of discovering this difficult.

3.1 Rules for relevancy

We suggest a method that does not change how the SAT solver works with respect to case-split heuristics, unit propagation, conflict resolution, etc. Instead, we convert to CNF using a variation of Tseitin algorithm, keep the input formula, and map every (Tseitin) auxiliary variable to a node in the original formula. Initially, only the auxiliary variable corresponding to the root in the original formula is marked as relevant. Relevancy is then propagated to sub-formulas using the following rules. Note how these rules effectively simulate the tableau rules. Assume φ is marked as relevant.

1. Let φ be shorthand for $\bigvee_i \varphi_i$, if φ is assigned true and is marked as relevant, then the first child φ_i that gets assigned true is marked relevant. If φ is assigned false and is marked as relevant, then all children are marked relevant.
2. Let φ be shorthand for $(\varphi_1 \leftrightarrow \varphi_2)$, if φ is marked as relevant, then both φ_1 and φ_2 are marked as relevant.
3. Let φ be $ite(\varphi_1, \varphi_2, \varphi_3)$, if φ is marked as relevant, then φ_1 is marked as relevant, and if φ_1 is assigned to true(false), then φ_2 (φ_3) is marked as relvant.

3.2 Implementing Relevancy

In our implementation the rules above are triggered during Boolean constraint propagation. The rules suggest that two different kind of events should be tracked: a variable is marked as relevant, a variable is assigned. Each variable has a *relevancy* bit attached to it. An *undo-list* is used to restore the value of this bit during backtracking. If a variable is a shorthand for some term, it also has the *term* attached to it. For each literal, a list *rw* of shorthands (variables) is also kept. The shorthand φ is a member of $rw[\varphi']$ iff $term[\varphi] = \varphi_1 \vee \ldots \vee \varphi_n$ and $\varphi' = \varphi_i$ for some $i \in [1, n]$, or $term[\varphi] = ite(\varphi', \varphi_2, \varphi_3)$. We say φ' is a child of φ. The lists *rw* are necessary because triggering rules 1 and 3 may depend on the truth assignment of a child variable (i.e., φ').

In standard DPLL(T), the atom attached to a Boolean variable φ is sent to the theory solver T as soon as φ is assigned by the SAT engine. When relevancy propagation is used, the atom is only sent to the theory solver T after φ is assigned and the relevancy bit is marked as true.

References

1. L. de Moura and N. Bjørner. Efficient E-matching for SMT Solvers. In *CADE'07*. Springer-Verlag, 2007.
2. H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast decision procedures. In *CAV 04: Computer Aided Verification*, LNCS 3114, pages 175–188, 2004.
3. G. S. Tseitin. On the complexity of derivation in propositional calculus. In *Automation of Reasoning 2: Classical Papers onComputational Logic 1967-1970*, pages 466–483. Springer-Verlag, 1983.