Application of SAT solvers to security:

Reps and Bryant et al. (2005) [1] use a bounded model checker called UCLID to check for exploits at the API level. UCLID translates the problem into one of checking the validity of a Boolean formula, which is checked using a SAT solver (CSP solver).

CSP solvers have many applications besides software security. We hope that our CSP Solver technology will also be useful in Computational Biology, such as pathway modeling based on planning and in Bayesian Inference.

Optimally Biased Coin:

Suppose that: for every variable in the formula, we flip a coin. If head, we set the variable to true otherwise we set it to false. If our coin is fair (i.e., it produces heads and tails with equal probability) then we can expect that half of the variables will be set to true and the other half will be set to false. However, this does not always lead to the best possible result.

We can improve this by biasing our coin (based on the formula) such that we can guarantee the maximum possible fraction of satisfied clauses. This fraction is called a P-optimal threshold because if anyone can guarantee a trillionth more, then P=NP. For example, for the 1in3 example described above, if we biased our coin so that it produces heads with probability 1/3 we guarantee that 4/9 of the clauses are guaranteed to be satisfied. The set of 1in3 problems where the fraction 4/9+trillionth can be satisfied is NP-complete. If we used a fair coin, only 3/8 of the clauses are guaranteed to be satisfied.

Derandomization:

Even if we are using a fair coin. We might flip our coin and get heads all the time. This means that we might be unable to satisfy the P-optimal threshold. Derandomization is about “guaranteeing” the P-optimal threshold deterministically.

Clause Learning:

Our goal is to satisfy all of the clauses not just the maximum fraction that can be set in polynomial time. Therefore, once we find out that a partial assignment lets one or more clauses unsatisfied, we add a new clause to the formula so that we never make the same mistake again. The following example demonstrates the learning process:

Step 1:  Step 2:  Step 3:  Step 4:

\[
\begin{align*}
\text{Decide } X_1 &= \text{false} \\
\text{Decide } X_2 &= \text{true} \\
\end{align*}
\]

Conflict reached via unit propagation: Learn or \((X_1 \& X_2)\)

Bitwise Relation Reduction:

Fast Unit Propagation for CSP: DPLL SAT solvers spend up to 90% [4] of their time doing unit propagation. Unit propagation for CSP is not so efficient. We have developed a representation scheme for relations, that allows us to do Unit Propagation for CSP in an extremely fast manner using bit-wise operations.

1. Our Approach:

   Clause learning:

   Bitwise relation reduction:

   4. Optimal biased coin: Universal P-optimal algorithm for fixed set of relations. [2] & [3] prove that if there is a better polynomial algorithm than using the optimally biased coin then P=NP.

2. Derandomization: explore with and without repeated optimization.


4. Bitwise relation reduction: experiment with bit-wise relation manipulation for CSP to speed up Unit Propagation, a very important operation in CSP solvers.

References:


Related to: Software Bounded Model Checker Boolean Formula SAT Solver