Verification using Satisfiability Checking, Predicate Abstraction, and Craig Interpolation

Himanshu Jain

THESIS ORAL TALK
Computer Systems are Pervasive

Computer Systems = Software + Hardware

Software/Hardware Bugs are common as well!
Building Reliable Computer Systems

Programming Languages
Testing
Static Analysis
Model Checking
Theorem Proving
Abstract Interpretation

Need for more automatic, scalable, precise verification tools
Decision Procedures

Reasoning engines in verification tools

Example: Boolean Satisfiability (SAT) Solvers
Decision Procedures++

Reasoning engines in verification tools

Example: Boolean Satisfiability (SAT) Solvers
Contributions of this thesis

Fast Non-clausal Boolean Satisfiability (SAT) Solvers
- Based on Negation Normal Form (NNF), hpgraph, vpgraph
- DPLL based algorithm
- General Matings based algorithm

Verifying Hardware at Register Transfer Level (RTL)
- Model checking RTL Verilog using Predicate Abstraction and Counterexample Guided Abstraction Refinement Loop

Efficient Craig Interpolation for Subsets of Integer Linear Arithmetic
- Integer linear equations, congruences, disequations
Contributions of this thesis

**Fast Non-clausal Boolean Satisfiability (SAT) Solvers**
- Based on Negation Normal Form (NNF), hpgraph, vpgraph
- DPLL based algorithm
- General Matings based algorithm

**Verifying Hardware at Register Transfer Level (RTL)**
- Model checking RTL Verilog using Predicate Abstraction and Counterexample Guided Abstraction Refinement Loop

**Efficient Craig Interpolation for Subsets of Integer Linear Arithmetic**
- Integer linear equations, congruences, disequations
Outline

- Motivation

- Fast Boolean Satisfiability (SAT) Solvers
  - DPLL Algorithm
  - Related work
  - Negation Normal Form, hpgraph, vpgraph
  - SAT solving using hpgraph, vpgraph

- Interpolation for Subsets of Integer Linear Arithmetic
  - An application

- Summary
**Boolean Satisfiability**

- **Applications in verification**
  - Model checking
  - Equivalence checking
  - Theorem proving
  - Test generation
  - Static analysis

- **Circuits in practice**
  - Thousands of inputs
  - Millions of gates
  - Structure sharing

---

### Boolean Circuit

- `^` for AND
- `v` for OR
- `¬a` for NOT a

Input variables: `a`, `b`
Davis-Putnam-Logemann-Loveland (DPLL) Algorithm

Input: Boolean Formula $\phi(x, y, z, \ldots)$
Find a satisfying assignment to $\phi$ by building a decision tree

A decision tree

Conflicts

Satisfying assignment

No need to explore
Most solvers apply DPLL on CNF

Clausal solvers: RSAT, PicoSat, MiniSat, BerkMin, zChaff, GRASP, .....

Boolean Circuit

Conjunctive Normal Form (CNF)

New variables introduced is proportional to number of gates
**Current SAT solvers**

- **Davis-Putnam-Logemann-Loveland (DPLL) algorithm**
  - Conversion of circuit to CNF

- **Number of variables proportional to number of gates**
  - Can be \(10^3-10^4\) times more than the number of inputs in a circuit
  - Slowdown due to large number of new variables (and clauses)

- **Modern CNF solvers use pre-processing techniques**
  - To reduce \#variables and \#clauses in CNF
  - But pre-processing has large memory requirements
Our SAT solving framework

- Boolean Circuit
- Negation Normal Form
- Requires introduction of new variables
- Fewer variables than CNF and more structure
- hpgraph
- vpgraph

DPLL algorithm
General Matings algorithm
Negation Normal Form (NNF)

An NNF formula contains arbitrary nesting of AND (\(\wedge\)) and OR (\(\vee\)) gates

- Negations can appear at leaf level
- No sharing of sub formulas
How to obtain NNF from Boolean Circuits

Circuit from converted to NNF by introducing one new variable \( y \) to remove sharing.
Satisfiability Checking of NNF Formulas

Our SAT algorithms will not add any more new variables
Why NNF: Number of variables

CNF without pre-processing has 5-10X more variables
Why NNF: Number of variables

Note no pre-processing for NNF
Our SAT solving framework

- Boolean Circuit
- Negation Normal Form
  - hpgraph
  - vpgraph

- Peter Andrew's Horizontal/Vertical Path Forms (1981)
- DPLL algorithm
- General Matings algorithm
Inductively creating \texttt{hpgraph(\phi)} from NNF formula \phi

\phi is a literal \(m\)

Create new node

\[ \phi = \phi_1 \land \phi_2 \]

Take graph union

\[ \phi = \phi_1 \lor \phi_2 \]

Add hyperedge from L_1 to R_2
Inductively creating \( \text{vpgraph}(\phi) \) from NNF formula \( \phi \)

Just interchange the handling of \( \land \) and \( \lor \)

\( \phi \) is a literal \( m \)

Create new node

\[ \phi = \phi_1 \land \phi_2 \]

Add hyperedge from \( L_1 \) to \( R_2 \)

\[ \phi = \phi_1 \lor \phi_2 \]

Take graph union
<table>
<thead>
<tr>
<th>Formula</th>
<th>Hpgraph</th>
<th>Vpgraph</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p )</td>
<td>( p )</td>
<td>( p )</td>
</tr>
<tr>
<td>( p \lor q )</td>
<td>( p \rightarrow q )</td>
<td>( p \lor q )</td>
</tr>
<tr>
<td>( (p \lor q) \land \neg r )</td>
<td>( p \rightarrow q )</td>
<td>( p \rightarrow q )</td>
</tr>
<tr>
<td>( (p \lor q) \land \neg r \land \neg q )</td>
<td>( p \rightarrow q )</td>
<td>( p \rightarrow q )</td>
</tr>
</tbody>
</table>
Example

Formula F: (((p ∨ q) ∧ ¬r ∧ ¬q) ∨ (¬p ∧ (r ∨ ¬s) ∧ q))
Example

Formula F: (((p ∨ q) ∧ ¬r ∧ ¬q) ∨ (¬p ∧ (r ∨ ¬s) ∧ q))

R denotes a root node and L denotes a leaf node
Representing NNF using Graphs

- Represent NNF formula $\phi$ in form of two graphs

- $\text{hpgraph}(\phi)$
  - Implicitly encodes all clauses present in $\text{CNF}(\phi)$

- $\text{vpgraph}(\phi)$
  - Implicitly encodes all cubes/terms present in $\text{DNF}(\phi)$

- Directed acyclic graphs, linear in the size of $\phi$

- **Horizontal/Vertical Path Forms [P. Andrews 1981]**
  - Used in higher order theorem proving (General Matings)
  - Not been explored for building fast SAT solvers
Our SAT solving framework

- Boolean Circuit
- Negation Normal Form
- hpgraph
- vpgraph

DPLL algorithm
General Matings algorithm
DPLL SAT solver

Boolean Constraint Propagation (BCP)
Boolean Constraint Propagation (BCP)

- Given a partial assignment $\sigma$ to variables in a formula $\phi$
- BCP determines if $\sigma$ falsifies $\phi$ (termed as conflict)
- Otherwise, BCP provides implied assignments (literals) due to $\sigma$

**CNF example:**

$$(a \lor b \lor c) \land
(\neg a \lor \neg b) \land
(\neg c \lor \neg a)$$

BCP detects a conflict when $\sigma := \{a=1, c=1\}$. For $\sigma := \{a=1\}$ BCP detects $\{c=0, b=0\}$ as implied literals
Efficient BCP in CNF SAT solvers

Two watched literal scheme [zChaff 2001]

Clause C: $l_1 \lor l_2 \lor l_3 \lor l_4 \lor l_5 \lor \ldots \lor l_{100} \lor \ldots$

- watch 1
- watch 2

$C$ is examined during BCP only when one of watches ($l_2$ or $l_5$) becomes false
DPLL on hpgraph
Meaning of BCP on hpgraph

Assignment $\sigma_1 := \{r=1, p=1\}$

Assignment $\sigma_2 := \{q=1, s=1\}$

Conflict clause: $\neg r \lor \neg p$

Unit clause: $\neg q \lor r \lor \neg s$

Implied literal: $r$
Can we generalize the CNF watched literal scheme?

- Watch two literals (nodes) on each clause (path)
- But exponential number of clauses (paths)!
Two Watched Cut Scheme

1. A node cut disconnects all horizontal paths
2. Watch two node cuts (allows observing two literals on each clause)
3. Minimal cuts (covered later)
1. Two cuts for each hpgraph component
2. Can be updated locally during BCP
3. Non-chronological backtracking is cheap
Our algorithm generalizes CNF watched literal scheme
Acceptable cut

A cut is acceptable if no literal appearing in it is false

Cut 1 is:

acceptable for

\( \sigma := \{ p=1 \} \)

Cut 2 is:

not acceptable for

\( \sigma := \{ p=1 \} \)

Now let us see the use of cuts during BCP
BCP Case 1: Both cuts are acceptable and disjoint

\( \sigma := \{s=1\} \)

No need to examine the hpgraph component

Intuitively, there will be no conflicts or implied literals
BCP Case 2 (conflict): No acceptable cut

An hpgraph has no acceptable cut if and only if current assignment falsifies the formula

\[ \neg q \lor \neg p \]

\[ \sigma := \{ p=1, q=1 \} \]
BCP Case 3 (Implications): Acceptable cuts but not disjoint

For $\sigma := \{p=1\}$ we can find two acceptable cuts.

We cannot find two completely node-disjoint cuts.

Intuitively nodes common to both cuts contain implied literals.

In our example $\neg r$ and $\neg q$ are implied literals.
BCP Case 3 (Implications): Acceptable cuts but not disjoint

Hpgraph component

\[ \sigma := \{p=1\} \]

\( \neg r \) is implied due to unit clause \( \neg r \lor \neg p \)

\( \neg q \) is implied due to unit clause \( \neg q \lor \neg p \)
Finding and Maintaining Minimal Cuts

hpgraph component

vpgraph component

minimal cut in hpgraph component = a path in vpgraph component
Finding and Maintaining Minimal Cuts

Depth first search for an acceptable path in vpgraph gives us an acceptable cut in hpgraph...
Overall Solver

1. Other features of modern SAT solvers

2. No pre-processing so far (circuit rewriting applicable)

Linear time

Generalized watched literal scheme
Experimental snapshot

- These techniques implemented in:
  - NFLSAT (Non-clausal Formulas SATisfiability checker)

- ~2500 non-clausal circuit formulas (industrial category)
  - Bounded model checking, k-induction, SW/HW verification
  - CNF obtained by adding new variables

- Comparing with state-of-the-art solvers
  - SAT-Race 2008 top two winners: MiniSAT++, Picoaigersat
  - Top three winners of SAT 2007 comp: RSAT, MiniSAT, PicoSAT
  - Using the competition binaries or latest source code
## Comparison with SAT 2007 winners

<table>
<thead>
<tr>
<th>Solver</th>
<th>Solved</th>
<th>Failed</th>
<th>Solved time</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFLSAT</td>
<td>2364</td>
<td>178</td>
<td>29753</td>
<td>131153</td>
</tr>
<tr>
<td>RSAT</td>
<td>2310</td>
<td>231</td>
<td>45794</td>
<td>184394</td>
</tr>
<tr>
<td>PicoSAT</td>
<td>2281</td>
<td>260</td>
<td>43298</td>
<td>199298</td>
</tr>
<tr>
<td>MiniSAT2</td>
<td>2272</td>
<td>269</td>
<td>50958</td>
<td>212358</td>
</tr>
<tr>
<td>MiniSAT2 070721</td>
<td>2270</td>
<td>271</td>
<td>39490</td>
<td>202089</td>
</tr>
</tbody>
</table>

Timeout 600 seconds per problem. Total 2541 problems.
NFLSAT vs. RSAT

NFLSAT solves 54 more problems

RSAT uses pre-processing
NFLSAT vs. PicoSAT

NFLSAT solves 83 more problems

PicoSAT does not use pre-processing
NFLSAT vs. MiniSAT2 070721

MiniSAT2 070721 uses pre-processing

NFLSAT solves 94 more problems
### Comparison SAT-Race 2008 winners And Inverter Graph (AIG) Track

<table>
<thead>
<tr>
<th>Solver</th>
<th>Solved</th>
<th>Failed</th>
<th>Solved time</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFLSAT</td>
<td>2060</td>
<td>132</td>
<td>26585</td>
<td>105785</td>
</tr>
<tr>
<td>MiniSAT++</td>
<td>2074</td>
<td>118</td>
<td>32457</td>
<td>103257</td>
</tr>
<tr>
<td>PicoaigerSAT</td>
<td>2033</td>
<td>159</td>
<td>35892</td>
<td>131292</td>
</tr>
</tbody>
</table>

Timeout 600 seconds per problem. Total 2192 problems.

If we remove 350 SAT 2007 benchmarks, NFLSAT (8504 sec) is twice as faster than MiniSAT++ (18119 sec)
NFLSAT vs. PicoaigerSAT

$\frac{\text{NFLSAT}}{\text{PicoaigerSAT}}$ (seconds)

SAT+UNSAT

$y > x$ on 283 pts

$y < x$ on 1711 pts
Outline

- Motivation

- Fast Boolean Satisfiability (SAT) Solvers
  - DPLL Algorithm
  - Related work
  - Negation Normal Form, hpgraph, vpgraph
  - SAT solving using hpgraph, vpgraph

- Interpolation for Subsets of Integer Linear Arithmetic
  - An application

- Summary
Decision Procedures++

Reasoning engines in verification tools

Formula -> Satisfiable

Formula -> Unsatisfiable (Craig Interpolants)
Interpolants [Craig 1957]

Given formulas F, G such that $F \land G$ is unsatisfiable

An interpolant for $(F,G)$ is a formula $I$:
1. $F \Rightarrow I$
2. $I \land G$ is unsatisfiable
3. $I$ contains only common variables of $F$ and $G$
Interpolants Example

Example 1 (propositional logic):
\[ F := p \land q \quad G := \neg q \land r \land s \]
\[ I := q \]

Example 2 (linear arithmetic):
\[ F := x + 2y \leq 3 \land -x - y \leq -1 \quad F \Rightarrow y \leq 2 \]
\[ G := y \geq 3 \]
\[ I := y \leq 2 \]
Interpolants in Verification [McMillan 2003]

Useful in symbolic model checking

Computing $\text{Reach}^1(S)$ requires existential quantification (costly using BDDs or SAT)
Interpolants in Verification [Jhala et al. 2004]

Useful for Property Directed Invariant Generation

Program $P$ \rightarrow \text{Predicate Abstraction} \rightarrow \text{Invariants for } P \text{ expressible in terms of } S

Predicates $S$ \rightarrow

Interpolants help in finding right set of predicates
How are Interpolants Obtained

\[ F \land G \quad \Rightarrow \quad \text{proof of unsatisfiability of } F \land G \quad \Rightarrow \quad F, G \]

Interpolant for \((F, G)\)
Existing Work on Computing Interpolants

Pudlak, McMillan, Jhala et al., Yorsh et al., Kapur et al., Rybalchenko et al., Kroening et al., Cimatti et al., Beyer et al.

- Can efficiently compute interpolants
  - For rational/real linear arithmetic
  - For equality with uninterpreted function symbols
  - Propositional logic (using SAT solvers)

- No efficient interpolation algorithms for
  - Integer linear arithmetic
  - Bit-vector arithmetic
  - Decision problem for conjunctions is itself NP-hard

We make progress in this direction.
Our results

- Polynomial time interpolation algorithms
  - For useful subsets of integer linear arithmetic

- Integer (Diophantine) linear equations
  - E.g. \( x = 3y \land 5x = 3z+u+2 \land ... \)

- Integer linear congruences (modular equations)
  - E.g. \( 4x = 2y + 9 \pmod{3} \land 2z + 5x -y = 7 \pmod{4} \land ... \)

- Integer linear equations and disequations
  - E.g. \( \neg(4x + 5y = 8) \land x = 3y \land ... \)
Interpolation for Integer Linear Equations

- F, G be conjunctions of integer linear equations

- We show that interpolant for (F,G) is always:
  - An integer linear equation or
  - An integer linear congruence

- F := \( x = 2y \) and G := \( x=2z+1 \)
  - An interpolant is \( x = 0 \pmod{2} \)
Interpolation Algorithm Step 1

Obtain a proof of unsatisfiability of $F \land G$
(How to get a contradiction from $F \land G$)

\[ F := (30 \times + 4y = 2) \]
\[ G := (y + 5z = 2) \]

\[ \frac{1}{5}, \frac{1}{5} \]

\[ \frac{1}{5} F + \frac{1}{5} G \] is equal to
\[ 6x + y + z = \frac{4}{5} \] (Contradiction)
Interpolation Algorithm Step 2

Sum the equations from F according to the proof of unsatisfiability

\[
\frac{1}{5} (30x + 4y = 2) + \frac{1}{5} (y + 5z = 2)
\]

\[
6x + \frac{4}{5}y = \frac{2}{5}
\]

We do not want \(x\)

Partial interpolant
Interpolation Algorithm Step 3

Remove variables not common to $F$ and $G$

\[ 6x + \frac{4}{5}y = \frac{2}{5} \]

\[ \frac{4}{5}y - \frac{2}{5} = -6x \]
\[ \Rightarrow \frac{4}{5}y - \frac{2}{5} \text{ is divisible by } 6 \]
\[ \Rightarrow \frac{4}{5}y - \frac{2}{5} = 0 \pmod{6} \]
\[ \Rightarrow 4y - 2 = 0 \pmod{30} \]

\[ 4y - 2 = 0 \pmod{30} \] is an interpolant for $(F, G)$
We have proved the correctness of above algorithm
Complexity of the Algorithm

- Obtain proof of unsatisfiability (step 1)
  - Polynomial time using Hermite Normal Form
  - Overall algorithm is polynomial time

- Multiple interpolants can be obtained
Multiple Interpolants

\[ G := y + 5z = 2 \]

\[ F := 30x + 4y = 2 \]

- \[ 4y - 2 \equiv 0 \pmod{5} \]
- \[ 4y - 2 \equiv 0 \pmod{10} \]
- \[ 4y - 2 \equiv 0 \pmod{15} \]
- \[ 4y - 2 \equiv 0 \pmod{30} \]
Outline

- Motivation

- Fast Boolean Satisfiability (SAT) Solvers
  - DPLL Algorithm
  - Related work
  - Negation Normal Form, hpgraph, vpgraph
  - SAT solving using hpgraph, vpgraph

- Interpolation for Subsets of Integer Linear Arithmetic
  - An application

- Summary
Predicate Discovery

```c
void main()
{
    int x=0, y=0;
    while(*)
    {
        x = x + 4*nondet();
        y = y + 8*nondet();
        assert(x+y != 1);
        assert(x+y != 2);
        assert(x+y != 3);
    }
}
```

Loop invariant:

x+y is divisible by 4
That is, x+y=0 (mod 4)

Such predicates can be found using our interpolation algorithms
Existing state-of-the-art tools such as BLAST, SATABs, VCEGAR cannot verify these programs. With the help of predicates found by our algorithms they can (VCEGAR).

<table>
<thead>
<tr>
<th>Example</th>
<th>Predicate</th>
<th>Time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex1</td>
<td>$y = 1 \mod 2$</td>
<td>2.72</td>
</tr>
<tr>
<td>ex2</td>
<td>$x + y = 0 \mod 2$</td>
<td>0.83</td>
</tr>
<tr>
<td>ex4</td>
<td>$x + y + z = 0 \mod 4$</td>
<td>0.95</td>
</tr>
<tr>
<td>ex5</td>
<td>$x = 0 \mod 4$, $y = 0 \mod 4$</td>
<td>1.1</td>
</tr>
<tr>
<td>ex6</td>
<td>$4x + 2y + z = 0 \mod 8$</td>
<td>0.93</td>
</tr>
<tr>
<td>ex7</td>
<td>$4x - 2y + z = 0 \mod 2^{22}$</td>
<td>0.54</td>
</tr>
<tr>
<td>forb1</td>
<td>$x + y = 0 \mod 3$</td>
<td>0.1</td>
</tr>
</tbody>
</table>
Outline

- **Motivation**

- **Fast Boolean Satisfiability (SAT) Solvers**
  - DPLL Algorithm
  - Related work
  - Negation Normal Form, hpgraph, vpgraph
  - SAT solving using hpgraph, vpgraph

- **Interpolation for Subsets of Integer Linear Arithmetic**
  - An application

- **Summary**
Summary

Boolean Circuit

Negation Normal Form

Non-Clausal SAT solver

vpgraph <-> hpgraph

BCP engine

Clause Database

Top-level DPLL Algorithm
Summary contd.

- **Efficient Interpolation Algorithms**
  - Integer linear equations
  - Integer linear congruences
  - Integer linear equations and disequations

- **Proofs of unsatisfiability**
  - Integer linear congruences
  - Integer linear equations and disequations
Thank you!

- **Edmund Clarke**
  - Exemplary *advisor*

- **Randy Bryant, Aarti Gupta, Ken McMillan, Frank Pfenning**
  - Excellent *thesis committee* members

- **Peter Andrews**
  - Math Logic classes and advice

- **Daniel Kroening, Orna Grumberg, Constantinos Bartzis**
  - Awesome *collaborators*

- **NEC verification group for great summer *internships***
Questions