SAT & QBF in Formal Verification

Armin Biere
Institute for Formal Models and Verification
Johannes Kepler University, Linz, Austria

RISC Seminar
Schloß Hagenberg
March 14, 2005
Overview

1. SAT
   - DPLL
   - Decision Heuristics and Learning

2. Bounded Model Checking

3. QBF
   - QBF for Symbolic Traversal
   - State-of-the-Art in QBF Solvers
   - Resolve & Expand
• input formula in conjunctive normal form (CNF)
  – a formula in **CNF** is a conjunction of clauses
  – each **clause** a disjunction of literals
  – a **literal** is positive ($v$) or negated boolean variable ($\neg v$)

\[
(\neg r \lor v) \land (s \lor v) \land (x \lor y \lor v) \land (\neg v \lor r) \land (\neg v \lor \neg x \lor \neg y \lor \neg r)
\]

• **SAT** = check whether formula in CNF is satisfiable
  (satisfiable = exists assignments which makes the formula true)
  – the NP complete problem
  – can be restricted (also in practice) to clauses of length 3
  – equivalent to check formula or circuit satisfiability
Tseitin Transformation: Circuit to CNF

equivalence checking problem

constraints

\[
\begin{align*}
& o \land (x \rightarrow a) \land (x \rightarrow c) \land (x \leftarrow a \land c) \land \ldots \\
& o \land (\bar{x} \lor a) \land (\bar{x} \lor c) \land (x \lor \bar{a} \lor \bar{c}) \land \ldots
\end{align*}
\]

implications

\[
\begin{align*}
& (x \leftrightarrow a \land c) \land \\
& (y \leftrightarrow b \lor x) \land \\
& (u \leftrightarrow a \lor b) \land \\
& (v \leftrightarrow b \lor c) \land \\
& (w \leftrightarrow u \land v) \land \\
& (o \leftrightarrow y \oplus w)
\end{align*}
\]

clauses
original clauses in which \( v \) or \( \neg v \) occurs:

\[
\begin{align*}
\neg r &\lor v \\
 s &\lor v \\
x \lor y \lor v
\end{align*}
\]

add non-trivial resolvents:

\[
(s \lor r), \quad (x \lor y \lor r), \quad \text{and} \quad (s \lor \neg x \lor \neg y \lor r)
\]

remove original clauses
Pure Literals

- pure literal $l$ in a CNF $f$
  - $l$ occurs in $f$
  - $\neg l$ does not occur in $f$

- clauses with pure literals can be removed
  - result $f\{l/1\}$
  - $f\{l/0\} \Rightarrow f\{l/1\}$

- stronger semantic criteria possible (e.g. autarkies)

- pure literal reduction as satisfiability preserving transformation
\[
\text{dp-sat()}
\]

\begin{verbatim}
forever
    boolean-constraint-propagation()
    if contains-empty-clause() then return unsatisfiable
    remove-clauses-with-pure-literals()
    if no-clause-left() then return satisfiable
    v := next-not-eliminated-variable()
    C_v := clauses-containing(v)
    C_{\neg v} := clauses-containing(\neg v)
    C' := \emptyset
    forall c_v \in C_v do
        forall c_{\neg v} \in C_{\neg v} do
            c' := resolve(v, c_v, c_{\neg v})
            if non-trivial(c') then C' := C' \cup \{c'\}
    replace C_v \cup C_{\neg v} by C'
\end{verbatim}
DPLL for SAT

[DavisLogemannLoveland62]

Trade Space for Time

\[
dpllsat(Assignment \ S)
\]

\[
\text{boolean-constraint-propagation()}
\]

\[
\text{if contains-empty-clause() then return unsatisfiable}
\]

\[
\text{if no-clause-left() then return satisfiable}
\]

\[
v := \text{next-unassigned-variable()}
\]

\[
\text{return } dpllsat(S \cup \{v \mapsto false\}) \lor dpllsat(S \cup \{v \mapsto true\})
\]

(pure literal rule omitted)
• early 90ies
  – focus on decision heuristics
  – 1st order heuristics
    * derived from current assignment plus formula
    * example: dynamic independent literal sum (DLIS)
    * does not take search history into account (⇒ 1st order)

• mid 90ies
  – non-chronological backtracking, learning, conflict driven assignment

Solvers: RELSAT, GRASP, SATO
Implication Graph and Learning

SAT: State-of-the-Art

level $n-2$

level $n-1$

level $n$

decision

conflict

learned clause: $(\neg v \lor \neg x \lor y \lor \neg z)$
Historical Perspective II

• end of 90ies
  – SAT solvers became mature enough to be used in various applications
    – e.g. in formal verification: bounded model checking (BMC)

• since 2000
  – wide spread industrial usage of SAT solvers in circuit verification
    – improved lazy data structures, 2nd order decision heuristics
    Solvers: ZCHAFF, BERKMIN
  – regular SAT solver competition
2nd Order Decision Heuristics

- take search history into account
  - focus on literals that recently contributed to conflicts
    1. increase score of literals in learned clauses
    2. exponentially decrease all scores over time
    3. pick unassigned variable with largest score
- works incredibly well in practice, but it is (still) unclear why

SAT & QBF in Formal Verification – RISC Seminar – March 2005
Armin Biere – JKU Linz
Explicit/Symbolic Checking

- model checking is about verifying temporal properties of systems algorithmically
  - builds on Pnueli’s idea on using temporal logic for specification purposes
  - explicit model checking represents states explicitly [EmersonClarke81]

- state explosion problem, particularly in hardware verification:
  - state space grows exponentially with the size of the system description
  - symmetry or partial order reduction as one solution

- symbolic model checking
  - symbolic representations for sets of states to combat the state explosion problem
    - originally with binary decision diagrams (BDDs)
      [CoudertMadre89,BurchClarkeMcMillanDillHwang90,McMillan93]
• **motivation:** leverage improvements of SAT technology for model checking
  
  – BDD based model checking did and does not scale as much as necessary
  
  – SAT seems to be more *robust* than BDDs

• **original idea:** shift focus towards falsification instead of verification
  
  – search for counter example traces of a certain length $k$
  
  – reformulate existence of a counter example of length $k$ as SAT problem

• **impact:**
  
  – industry uses simulation, then bounded and finally BDD based model checking
  
  – accelerated interest in SAT technology
checking safety property $G_p$ for a bound $k$ as SAT problem:

$I(s_0) \land T(s_0, s_1) \land \cdots \land T(s_{k-1}, s_k) \land \bigvee_{i=0}^{k} \neg p(s_i)$

check occurrence of $\neg p$ in the first $k$ states
generic counter example trace of length $k$ for liveness $\mathbf{F}p$

$I(s_0) \wedge T(s_0,s_1) \wedge \cdots \wedge T(s_k,s_{k+1}) \wedge \bigvee_{i=0}^{k} s_i = s_{i+1} \wedge \bigwedge_{i=0}^{k} \neg p(s_i)$

(however we recently showed that liveness can always be reformulated as safety [BiereArthoSchuppan02])
find bounds on the maximal length of counter examples
  – also called **completeness threshold**
  – exact bounds are hard to find ⇒ approximations

-- induction
  – use of inductive invariants (manually generated)
  – generalization of inductive invariants: **pseudo induction** or \( k \)-induction

-- use SAT for quantifier elimination as with BDDs
  – then model checking becomes fixpoint calculation
  – alternatively use approximate elimination (as in McMillan’s interpolation)

-- or in an abstraction/refinement loop
Symbolic Transitive Closure

\[ T \] boolean formula encoding of a (finite transition) relation
\[ [[T]] \subseteq \{0, 1\}^n \times \{0, 1\}^n \]

Transitive Closure

\[ T^* \equiv T^{2n} \]

Standard Linear Unfolding

\[ T^{i+1}(s, t) \equiv \exists m. T^i(s, m) \land T(m, t) \]

Iterative Squaring via Copying

\[ T^{2 \cdot i}(s, t) \equiv \exists m. T^i(s, m) \land T^i(m, t) \]

Non Copying Iterative Squaring

\[ T^{2 \cdot i}(s, t) \equiv \exists m. \forall c. \exists l, r. (c \rightarrow (l, r) = (s, m)) \land (\overline{c} \rightarrow (l, r) = (m, t)) \land T^i(l, r) \]
DPLL for SAT and QBF

**dpll-sat**(*Assignment S*)

boolean-constraint-propagation()

if contains-empty-clause() then return false

if no-clause-left() then return true

\( \nu := \text{next-unassigned-variable}() \)

return \( \text{dpll-sat}(S \cup \{\nu \mapsto false\}) \lor \text{dpll-sat}(S \cup \{\nu \mapsto true\}) \)

**dpll-qbf**(*Assignment S*)

boolean-constraint-propagation()

if contains-empty-clause() then return false

if no-clause-left() then return true

\( \nu := \text{next-outermost-unassigned-variable}() \)

\( \& := \text{is-existential}(\nu) \lor \land \)

return \( \text{dpll-sat}(S \cup \{\nu \mapsto false\}) \land \text{dpll-sat}(S \cup \{\nu \mapsto true\}) \)
Why is QBF harder than SAT?

\[\models \forall x . \exists y . (x \leftrightarrow y)\]

\[\not\models \exists y . \forall x . (x \leftrightarrow y)\]

Decision Order Matters!
State-of-the-Art

QBF: Resolve & Expand

- almost all implementations are QBF-enhanced DPLL: [Cadoli...98] [Rintanen01]
- recently learning was added [Giunchiglia...01] [Letz01] [ZhangMalik02]
- all deterministic solvers (except one) in QBF-Evaluation’03 were DPLL based
  - top-down: split on variables from the outside to the inside

- multiple quantifier elimination procedures:
  - enumeration [PlaistedBiereZhu03] [McMillan02]
  - expansion [Aziz-Abdulla...00] [WilliamsBiere...00] [AyariBasin02]
    - bottom-up: eliminate variables from the inside to the outside

- q-resolution [Kleine-Büning...95]
Forall Reduction and Q-Resolution

- **collect** variables in scopes, **order** variables and scopes according to nesting depth:

\[
\begin{align*}
\exists a, b, c, d. \\
\forall x, y, z. \\
\exists r, s, t. & \quad (c \lor d)(a \lor \bar{c} \lor \bar{x} \lor y)(\bar{a} \lor x \lor s)(t \lor \ldots) & \cdots \\
\text{scope 0} & \quad \text{scope 1} & \quad \text{scope 2}
\end{align*}
\]

- **attach** clauses to the scope of its innermost variables

- **remove** innermost universal literals in clauses attached to universal scopes:

\[
(a \lor \bar{c} \lor \bar{x} \lor y) \quad \text{simplifies to} \quad (a \lor \bar{c})
\]

- q-resolution = resolution + forall reduction
all clauses are forall reduced

⇒ innermost scope is always existential
⇒ no clauses attached to universal scopes

normalized structure of quantified CNF:

$$\Omega(S_1) \cdot S_1 \cdot \Omega(S_2) \cdot S_2 \cdot \ldots \cdot \forall S_{m-1} \cdot \exists S_m \cdot f \land g \quad m \geq 2$$

$$f \equiv \text{clauses of scope } S_m$$

$$g \equiv \text{clauses of outer scopes} \quad S_i, \quad i < m - 1$$

$$S_{\exists} \equiv S_m$$

$$S_{\forall} \equiv S_{m-1}$$
Algorithm

\textit{QBF: Resolve & Expand}

\begin{verbatim}
resolve-and-expand()

forever

simplify()

if contains-empty-clause() then return false

if no-clause-left() then return true

if is-propositional() then return sat-solve(0)

v := schedule-cheapest-to-eliminate-variable()

if is-existential(v) then resolve(v)

if is-universal(v) then expand(v)
\end{verbatim}
original clauses in which \( v \) or \( \neg v \) occurs:

\[
\begin{align*}
\neg r \lor v \\
\neg r \lor v \\
x \lor y \lor v
\end{align*}
\]

add \textbf{forall reduced} non-trivial resolvents:

\[
( s \lor r ), \quad (x \lor y \lor r), \quad \text{and} \quad (s \lor \neg x \lor \neg y \lor r)
\]

remove original clauses
one-to-one mapping of variables: \( u \in S_\exists \) mapped to \( u' \in S'_\exists \)

before expansion:

\[
\Omega(S_1) S_1 . \quad \Omega(S_2) S_2 . \quad \ldots \quad \forall S_A \quad . \quad \exists S_\exists \quad . \quad f \land g
\]

after expansion:

\[
\Omega(S_1) S_1 . \quad \Omega(S_2) S_2 . \quad \ldots \quad \forall (S_A - \{v\}) . \quad \exists (S_\exists \cup S'_\exists) . \quad f\{v/0\} \land f'\{v/1\} \land g
\]
• elimination cost: \( o(l) \) \equiv \text{number of expected added literals} \\
\( s(l) \) \equiv \text{sum of lengths of clauses with literal } l \\
\( s(S) \) \equiv \text{sum lengths of clauses with scope } S \\

• expansion cost: \( s(S_{\exists}) = \left(s(v) + s(\neg v) + o(v) + o(\neg v)\right) \)

• resolution cost: \( o(\neg v) \cdot \left(s(v) - o(v)\right) + o(v) \cdot \left(s(\neg v) - o(\neg v)\right) - \left(s(v) + s(\neg v)\right) \)
<table>
<thead>
<tr>
<th>benchmark family</th>
<th>#inst</th>
<th>decide</th>
<th>qube</th>
<th>semprop</th>
<th>expand</th>
<th>quantor</th>
</tr>
</thead>
<tbody>
<tr>
<td>adder*</td>
<td>16</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Adder2*</td>
<td>14</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>C[0-9]*</td>
<td>27</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>CHAIN*</td>
<td>11</td>
<td>10</td>
<td>7</td>
<td>11</td>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>comp*</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>flip*</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>impl*</td>
<td>16</td>
<td>12</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>k*</td>
<td>171</td>
<td>77</td>
<td>91</td>
<td>97</td>
<td>60</td>
<td>108</td>
</tr>
<tr>
<td>mutex*</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>robots*</td>
<td>48</td>
<td>0</td>
<td>36</td>
<td>36</td>
<td>15</td>
<td>24</td>
</tr>
<tr>
<td>term1*</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>toilet*</td>
<td>260</td>
<td>187</td>
<td>260</td>
<td>260</td>
<td>259</td>
<td>259</td>
</tr>
<tr>
<td>TOILET*</td>
<td>8</td>
<td>8</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>tree*</td>
<td>12</td>
<td>10</td>
<td>12</td>
<td>12</td>
<td>8</td>
<td>12</td>
</tr>
</tbody>
</table>

#(among best in family)

#(single best in family)

(families with no difference and two actually random families removed)
Simplify

- resolve quadratic in number of occurrences, expand may double the size
  \[ \Rightarrow \text{simplify CNF as much as possible before elimination} \]

- standard simplification: **unit propagation, pure literal rule, forall reduction**

- **equivalence reasoning**: extract bi-implications and substitute variables
  \[ \forall x . \exists y . (x \lor y) (x \rightarrow y) (y \rightarrow x) \equiv \forall x . \exists y . (x \lor y) (x = y) \equiv \forall x . \exists y . (x \lor x) \equiv 0 \]

- **subsumption**: remove subsumed clauses
  - backward subsumption is checked on-the-fly whenever a clause is added
  - forward subsumption is expensive and only checked before expensive operations
<table>
<thead>
<tr>
<th>hard instance</th>
<th>time</th>
<th>space</th>
<th>∀</th>
<th>∃</th>
<th>units</th>
<th>pure</th>
<th>subsu.</th>
<th>subst.</th>
<th>∀red.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Adder2-6-s</td>
<td>29.6</td>
<td>19.7</td>
<td>90</td>
<td>1618</td>
<td>13732</td>
<td>126</td>
<td>13282</td>
<td>174081</td>
<td>0 37268</td>
</tr>
<tr>
<td>2 adder-4-sat</td>
<td>0.2</td>
<td>2.8</td>
<td>42</td>
<td>1618</td>
<td>13926</td>
<td>0</td>
<td>884</td>
<td>6487</td>
<td>0 960</td>
</tr>
<tr>
<td>3 adder-6-sat</td>
<td>36.6</td>
<td>22.7</td>
<td>90</td>
<td>1618</td>
<td>13926</td>
<td>0</td>
<td>7290</td>
<td>197091</td>
<td>0 54174</td>
</tr>
<tr>
<td>4 C49<em>1.</em>_0_0*</td>
<td>27.9</td>
<td>13.3</td>
<td>2</td>
<td>579</td>
<td>2288</td>
<td>10</td>
<td>48</td>
<td>4552</td>
<td>84 2494</td>
</tr>
<tr>
<td>5 C5<em>1.</em>_0_0*</td>
<td>56.2</td>
<td>16.0</td>
<td>2</td>
<td>977</td>
<td>1087</td>
<td>72</td>
<td>85</td>
<td>3242</td>
<td>2 725</td>
</tr>
<tr>
<td>6 k_path_n-15</td>
<td>0.1</td>
<td>0.8</td>
<td>32</td>
<td>1042</td>
<td>1146</td>
<td>76</td>
<td>106</td>
<td>4470</td>
<td>2 909</td>
</tr>
<tr>
<td>7 k_path_n-16</td>
<td>0.1</td>
<td>0.8</td>
<td>34</td>
<td>1042</td>
<td>1146</td>
<td>76</td>
<td>106</td>
<td>4470</td>
<td>2 909</td>
</tr>
<tr>
<td>8 k_path_n-17</td>
<td>0.1</td>
<td>0.9</td>
<td>36</td>
<td>1087</td>
<td>1240</td>
<td>84</td>
<td>149</td>
<td>3967</td>
<td>2 855</td>
</tr>
<tr>
<td>9 k_path_n-18</td>
<td>0.1</td>
<td>0.9</td>
<td>36</td>
<td>1146</td>
<td>1318</td>
<td>84</td>
<td>130</td>
<td>4470</td>
<td>2 909</td>
</tr>
<tr>
<td>10 k_path_n-20</td>
<td>0.1</td>
<td>0.9</td>
<td>38</td>
<td>1240</td>
<td>1318</td>
<td>84</td>
<td>130</td>
<td>4470</td>
<td>2 909</td>
</tr>
<tr>
<td>11 k_path_n-21</td>
<td>0.1</td>
<td>1.0</td>
<td>40</td>
<td>1318</td>
<td>1318</td>
<td>84</td>
<td>130</td>
<td>4470</td>
<td>2 909</td>
</tr>
<tr>
<td>12 k_t4p_n-7</td>
<td>15.5</td>
<td>105.8</td>
<td>43</td>
<td>88145</td>
<td>138</td>
<td>193</td>
<td>63876</td>
<td>760844</td>
<td>8 215</td>
</tr>
<tr>
<td>13 k_t4p_p-8</td>
<td>5.8</td>
<td>178.6</td>
<td>29</td>
<td>12798</td>
<td>130136</td>
<td>193</td>
<td>63876</td>
<td>938973</td>
<td>4 137</td>
</tr>
<tr>
<td>14 k_t4p_p-9</td>
<td>0.3</td>
<td>4.5</td>
<td>32</td>
<td>4179</td>
<td>137</td>
<td>1389</td>
<td>23344</td>
<td>1499430</td>
<td>4 140</td>
</tr>
<tr>
<td>15 k_t4p_p-10</td>
<td>27.9</td>
<td>152.9</td>
<td>35</td>
<td>130136</td>
<td>193</td>
<td>63876</td>
<td>938973</td>
<td>4 137</td>
<td></td>
</tr>
<tr>
<td>16 k_t4p_p-11</td>
<td>86.0</td>
<td>471.5</td>
<td>38</td>
<td>196785</td>
<td>204</td>
<td>79547</td>
<td>1499430</td>
<td>4 140</td>
<td></td>
</tr>
<tr>
<td>17 k_t4p_p-15</td>
<td>84.6</td>
<td>354.7</td>
<td>50</td>
<td>240892</td>
<td>169</td>
<td>181676</td>
<td>1336774</td>
<td>9 226</td>
<td></td>
</tr>
<tr>
<td>18 k_t4p_p-20</td>
<td>3.6</td>
<td>16.1</td>
<td>65</td>
<td>27388</td>
<td>182</td>
<td>21306</td>
<td>197273</td>
<td>11 325</td>
<td></td>
</tr>
</tbody>
</table>

time in seconds, space in MB
<table>
<thead>
<tr>
<th>hard instance</th>
<th>time (seconds)</th>
<th>space (MB)</th>
<th>∀</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adder2-6-s</td>
<td>12.2</td>
<td>m.o.</td>
<td>–</td>
</tr>
<tr>
<td>adder-4-sat</td>
<td>12.1</td>
<td>m.o.</td>
<td>–</td>
</tr>
<tr>
<td>adder-6-sat</td>
<td>13.0</td>
<td>m.o.</td>
<td>–</td>
</tr>
<tr>
<td>C49<em>1.</em>_0_0*</td>
<td>98.3</td>
<td>40.8</td>
<td>1</td>
</tr>
<tr>
<td>C5<em>1.</em>_0_0*</td>
<td>357.0</td>
<td>45.6</td>
<td>2</td>
</tr>
<tr>
<td>k_path_n-15</td>
<td>16.5</td>
<td>m.o.</td>
<td>–</td>
</tr>
<tr>
<td>k_path_n-16</td>
<td>16.6</td>
<td>m.o.</td>
<td>–</td>
</tr>
<tr>
<td>k_path_n-17</td>
<td>16.2</td>
<td>m.o.</td>
<td>–</td>
</tr>
<tr>
<td>k_path_n-18</td>
<td>16.8</td>
<td>m.o.</td>
<td>–</td>
</tr>
<tr>
<td>k_path_n-20</td>
<td>21.4</td>
<td>m.o.</td>
<td>–</td>
</tr>
<tr>
<td>k_path_n-21</td>
<td>21.0</td>
<td>m.o.</td>
<td>–</td>
</tr>
<tr>
<td>k_t4p_n-7</td>
<td>16.8</td>
<td>m.o.</td>
<td>–</td>
</tr>
<tr>
<td>k_t4p_p-8</td>
<td>21.4</td>
<td>m.o.</td>
<td>–</td>
</tr>
<tr>
<td>k_t4p_p-9</td>
<td>21.2</td>
<td>m.o.</td>
<td>–</td>
</tr>
<tr>
<td>k_t4p_p-10</td>
<td>17.3</td>
<td>m.o.</td>
<td>–</td>
</tr>
<tr>
<td>k_t4p_p-11</td>
<td>17.3</td>
<td>m.o.</td>
<td>–</td>
</tr>
<tr>
<td>k_t4p_p-15</td>
<td>21.3</td>
<td>m.o.</td>
<td>–</td>
</tr>
<tr>
<td>k_t4p_p-20</td>
<td>20.9</td>
<td>m.o.</td>
<td>–</td>
</tr>
</tbody>
</table>

time in seconds, space in MB, m.o. = memory out (> 1 GB)