Preprocessing CNF instances: SatELite

Albert Oliveras and Enric Rodríguez-Carbonell

Logic and Algebra in Computer Science
Session 5
Fall 2009, Barcelona
Overview of the session

- Why should we preprocess?

  - Empirical observations:
    - Clause distribution
    - Self-subsuming resolution
    - Var. elimination by substitution

  - Overall algorithm

  - Demo and experimental results
Motivation

SAT solvers are successfully used in a very different areas, but:
- CNF conversion is usually done by Tseitin transformation
- Application-dependent smarter encodings work better
- SAT users only want to be users, not developers!

Two possible solutions:
- Develop smart CNF conversions
- Preprocess already converted CNF formulas

Here we take the second solucion: PREPROCESS
Preprocess

- GOAL: convert the CNF formula into a better one

- What does “better” mean?
  - **Smaller**
    - In general, size $\neq$ difficulty of a formula
    - Among similarly generated formulas, smaller $\approx$ easier
  - **Better-suited** for SAT solver
    - SAT solvers’ only deduction rule is unit propagation
    - We should try to make unit propagation more powerful

- We will focus on the **SatELite** preprocessor:
  - Light-weight approach (not too much time preprocessing)
  - Focus is on reducing size (size = number of clauses)
Resolution, again...

\[
\begin{align*}
\frac{p \lor C \quad \neg p \lor D}{C \lor D}
\end{align*}
\]

- Given clause set \( S \) we can:
  - Choose a variable \( p \in S \)
  - Let \( S_p = \{ p \lor C \mid p \lor C \in S \text{ and is not a tautology} \} \)
  - Let \( S_{\overline{p}} = \{ \neg p \lor D \mid \neg p \lor D \in S \text{ and is not a tautology} \} \)
  - Take \( S_p \otimes S_{\overline{p}} := \{ C \lor D \mid p \lor C \in S_p \text{ and } \neg p \lor D \in S_{\overline{p}} \} \)
  - The clause set \( S \setminus \{ S_p \cup S_{\overline{p}} \} \cup S_p \otimes S_{\overline{p}} \)
    - Contains one variable less than \( S \)
    - Is equisatisfiable to \( S \).

- If we iterate the process we get a decision procedure for SAT
  (\( \approx \) original Davis-Putnam algorithm [’60])

- Problem: clauses sets may grow too much

Preprocessing CNF instances: SatELite – p. 5
Resolution, again (2)

Remember \( S_p \otimes S_{\overline{p}} := \{ C \lor D \mid p \lor C \in S_p \text{ and } \neg p \lor D \in S_{\overline{p}} \} \)

**QUESTION:** Why \( S' := S \setminus \{S_p \cup S_{\overline{p}}\} \cup S_p \otimes S_{\overline{p}} \) is equisatisfiable to \( S \)?

**Proof sketch:**

\( S \) satisf. \( \Rightarrow \) \( S' \) satisf. is trivial (resolution correct)

\( S' \) satisf. \( \Rightarrow \) \( S \) satisf.?

Take \( I \) model of \( S' \) and extend it to \( p \)

\( I(p) = 0 \) iff \( I \not| D \) for some \( \neg p \lor D \) in \( S_{\overline{p}} \)

Now obviously \( I \models S_{\overline{p}} \)

If \( I \not| S_p \), there is \( p \lor C \in S_p \) with \( I \not| p \lor C \)

Necessarily \( I(p) = 0 \), and hence there is a clause \( \neg p \lor D \in S_{\overline{p}} \) with \( I \not| D \). But then clause \( C \lor D \in S' \) and \( I \not| C \lor D \). Contradiction!

**IMPORTANT:** effective extension of models if \( p \) is eliminated
Overview of the session

- Why should we preprocess?

- **Empirical observations:**
  - Clause distribution
  - Self-subsuming resolution
  - Var. elimination by substitution

- Overall algorithm

- Demo and experimental results
Observation 1 - Clause Distribution

The clause set $S \otimes S'$ is called the **clause distribution** of $S$ and $S'$

Exhaustive application of clause distribution is too productive

But.....

- Removing some vars might decrease number of clauses

- **Empirical observation:**

  Clause distribution generates lots of subsumed clauses

  ([Def.] $C$ subsumes $C'$ iff $C \subseteq C'$)

**EXAMPLE:**

$S := \{q \lor s \lor r, \ p \lor q \lor \neg t \lor r, \ \neg p \lor q \lor s\}$

After removing $p$ we would obtain the clause set

$\{q \lor s \lor r, \ q \lor \neg t \lor r \lor s\}$

but the added clause is subsumed by the first one
Observation 2 – Self-subsuming Resolution

- Consider the clauses \( x \lor a \lor b \) and \( \neg x \lor a \)

- 2nd clause almost subsumes 1st, but \( x \) has different polarity

- Consider the resolution step

\[
\begin{array}{c}
  x \lor a \lor b \\
  \neg x \lor a \\
  \hline
  a \lor b
\end{array}
\]

- Conclusion subsumes first premise hence we remove premise

- \( x \lor a \lor b \) has been strengthened by self-subsuming resolution

- Note that it allows us to reduce the size of an existing clause
Obser. 3 - Variable elimination by substitution

Most CNF instances are obtained after Tseitin conversion

Hence, it is easy to identify and/or definitions:

- $\overline{x} \lor a \lor b$, $x \lor \overline{a}$, $x \lor \overline{b}$ is $x \leftrightarrow a \lor b$
- $x \lor \overline{a} \lor \overline{b}$, $\overline{x} \lor a$, $\overline{x} \lor b$ is $x \leftrightarrow a \land b$

Consider $S := \{x \lor c, x \lor \overline{d}, x \lor \overline{a} \lor \overline{b}, \overline{x} \lor a, \overline{x} \lor b, \overline{x} \lor \overline{e} \lor \overline{f}\}$

If we try to remove $x$ by clause distribution we get:

$c \lor a$, $c \lor b$, $\overline{d} \lor a$, $\overline{d} \lor b$, $\overline{a} \lor \overline{b} \lor \overline{e} \lor \overline{f}$ $(R_x \otimes \overline{G_x} \cup R_{\overline{x}} \otimes G_x)$

$\overline{a} \lor \overline{b} \lor a$, $\overline{a} \lor \overline{b} \lor b$ $(G_x \otimes \overline{G_x})$ $c \lor \overline{e} \lor \overline{f}$, $\overline{d} \lor \overline{e} \lor \overline{f}$ $(R_x \otimes R_{\overline{x}})$

We observe:

- $G_x \otimes \overline{G_x}$ only contains tautologies
- $R_x \otimes \overline{G_x} \cup R_{\overline{x}} \otimes G_x \models R_x \otimes R_{\overline{x}}$

Hence we replace $S$ (6 clauses) by $R_x \otimes \overline{G_x} \cup R_{\overline{x}} \otimes G_x$ (5 clauses)
Overview of the session

- Why should we preprocess?

- Empirical observations:
  - Clause distribution
  - Self-subsuming resolution
  - Var. elimination by substitution

- Overall algorithm

- Demo and experimental results
Overall Preprocessing Algorithm

do
  do
    for each $C \in S$ do selfSubsumeWith($C$)
    unitPropagate()
  while (someProgress)

for each $C \in S$ do subsumeWith($C$)

do
  for each $x \in S$ do tryToEliminate($x$)
  while (someProgress)
while (someProgress)
Pending things:

- How to implement:
  - selfSubsumeWith($C$)
  - subsumeWith($C$)
  - tryToEliminate($x$)

- Refine the algorithm so that: [read paper for details]
  - someProgress is clearly defined
  - Not all clauses and variables are tried every time
Subsumption and Self-subsumption

```plaintext
subsumeWith (Clause C) returns SetOfClauses
   // returns all clauses in S subsumed by C
   pick literal l ∈ C with the shortest occurList
   for each C' ∈ occurList(l) do
      if C ≠ C' and subset(C, C') then
         add C' to result
   return result

selfSubsumeWith (Clause C)
   for each l ∈ C do
      for each C' ∈ subsumeWith(C[l := ¬l]) do
         remove ¬l from C'
```

Preprocessing CNF instances: SatELite – p. 12
Subsumption and Self-subsumption

subsumeWith (Clause C) returns SetOfClauses
   // returns all clauses in S subsumed by C
   pick literal \( l \in C \) with the shortest occurList
   for each \( C' \in \text{occurList}(l) \) do
      if \( C \neq C' \) and \( \text{subset}(C,C') \) then
         add \( C' \) to result
   return result

selfSubsumeWith (Clause C)
   for each \( l \in C \) do
      for each \( C' \in \text{subsumeWith}(C[l:=\neg l]) \) do
         remove \( \neg l \) from \( C' \)
Subsumption and Self-subsumption (2)

How to implement $\text{subset}(C, C') = C$ is a subset of $C'$?

- Each clause has its size and a 64-bit signature:
  - Take a hash function $h : \text{Literals} \rightarrow [0...63]$
  - The signature of $C$ is the bitwise-or of $2^{h(l)}$ for each $l \in C$
  - Clearly, if $C \subseteq C'$ then $\text{sig}(C)$ bitwise implies $\text{sig}(C')$

```
subset(Clause C, Clause C')
// PRECONDITION: C \neq C'
// returns whether C is a subset of C'
if size(C) \geq size(C') then return FALSE
if \text{sig}(C) \text{ does not bitwise imply } \text{sig}(C') then return FALSE
return complete (expensive) check C \subseteq C'
```
Variable elimination

```
tryToEliminate (Variable x)
    // tries to eliminate variable x
    if x has zero occurrences then return
    if occurList(x) > 10 and occurList(\bar{x}) > 10 then return

    // will remove x only if this reduces num. of clauses
    def = findDefinition(x)
    if def ≠ NO_DEF then tryVarSubstitution(def)
    else tryClauseDistribute(def)

    if x was eliminated then unitPropagate()
```

- Note that expensive things are not even tried
- 10 is a heuristic cut-off value
Overview of the session

- Why should we preprocess?

- Empirical observations:
  - Clause distribution
  - Self-subsuming resolution
  - Var. elimination by substitution

- Overall algorithm

- Demo and experimental results
Bibliography - Some further reading

Paper:


Other resources:

- [http://minisat.se/SatELite.html](http://minisat.se/SatELite.html)