Preprocessing CNF instances: SatELite

Albert Oliveras and Enric Rodríguez-Carbonell

Logic and Algebra in Computer Science

Session 5

Fall 2009, Barcelona

Departament de Llenguatges i Sistemes Informatics

UNIVERSITAT POLITECNICA DE CATALUNYA

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

Departament de Llenguatges i Sistemes Informatics
UNIVERSITAT POLITÈCNICA DE CATALUNYA

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...

$$\frac{p \vee C \quad \neg p \vee D}{C \vee D}$$

- Given clause set S we can:
  - Choose a variable $p \in S$
  - Let $S_p = \{\, p \vee C \mid p \vee C \in S$ and is not a tautology $\}$
  - Let $S_{\overline{p}} = \{\, \neg p \vee D \mid \neg p \vee D \in S$ and is not a tautology $\}$
  - Take $S_p \otimes S_{\overline{p}} := \{ C \vee D \mid p \vee C \in S_p$ and $\neg p \vee D \in S_{\overline{p}} \}$
  - 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

Resolution, again (2)

Remember $S_p \otimes S_{\overline{p}} := \{C \vee D \mid p \vee C \in S_p \text{ and } \neg p \vee D \in S_{\overline{p}}\}$

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 $I \not\models D$ for some $\neg p \vee D$ in $S_{\overline{p}}$

Now obviously $I \models S_{\overline{p}}$

If $I \not\models S_p$, there is $p \vee C \in S_p$ with $I \not\models p \vee C$

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

Departament de Llenguatges i Sistemes Informatics
UNIVERSITAT POLITÈCNICA DE CATALUNYA

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:

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

    After removing p we would obtain the clause set

    $\{q \vee s \vee r,\ \ \mathbf{q} \vee \neg \mathbf{t} \vee \mathbf{r} \vee \mathbf{s}\}$

    but the added clause is subsumed by the first one

Observation 2 – Self-subsuming Resolution

- Consider the clauses $x \vee a \vee b$ and $\neg x \vee a$
- 2nd clause almost subsumes 1st, but x has different polarity
- Consider the resolution step

$$\frac{x \vee a \vee b \quad \neg x \vee a}{a \vee b}$$

- Conclusion subsumes first premise hence we remove premise
- $x \vee a \vee b$ has been strengthed 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:
  - $\overline{x} \vee a \vee b,\ \ x \vee \overline{a},\ \ x \vee \overline{b}$ is $x \leftrightarrow a \vee b$
  - $x \vee \overline{a} \vee \overline{b},\ \ \overline{x} \vee a,\ \ \overline{x} \vee b$ is $x \leftrightarrow a \wedge b$
- Consider $S := \{\underbrace{x \vee c,\ \ x \vee \overline{d}}_{R_x},\ \underbrace{x \vee \overline{a} \vee \overline{b}}_{G_x},\ \underbrace{\overline{x} \vee a,\ \ \overline{x} \vee b}_{G_{\overline{x}}},\ \underbrace{\overline{x} \vee \overline{e} \vee f}_{R_{\overline{x}}}\}$

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

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

  $\overline{a} \vee \overline{b} \vee a,\ \ \overline{a} \vee \overline{b} \vee b\ \ (G_x \otimes G_{\overline{x}}) \qquad c \vee \overline{e} \vee f,\ \ \overline{d} \vee \overline{e} \vee f\ \ (R_x \otimes R_{\overline{x}})$
- We observe:
  - $G_x \otimes G_{\overline{x}}$ only contains tautologies
  - $R_x \otimes G_{\overline{x}} \cup R_{\overline{x}} \otimes G_x \ \models\ R_x \otimes R_{\overline{x}}$
- Hence we replace S (6 clauses) by $R_x \otimes G_{\overline{x}} \cup R_{\overline{x}} \otimes G_x$ (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

Departament de Llenguatges i Sistemes Informatics
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Overall Preprocessing Algorithm

```
do
    do
        for each C ∈ S do selfSubsumeWith(C)
        unitPropagate()
    while(someProgress)
    for each C ∈ S do subsumeWith(C)
    do
        for each x ∈ S do tryToEliminate(x)
    while(someProgress)
while(someProgress)
```

Overall Preprocessing Algorithm (2)

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

```
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 := l̄]) do
            remove l̄ from C'
```

Subsumption and Self-subsumption

```
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 := l̄]) do
            remove l̄ from C'
```

Subsumption and Self-subsumption (2)

How to implement $subset(C, C')$ = C is a subset of C' ?

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

```
subset(Clause C, Clause C')
    // PRECONDITION: C ≠ C'
    // returns whether C is a subset of C'

    if size(C) ≥ size(C') then return FALSE
    if sig(C) does not bitwise imply sig(C') then return FALSE
    return complete (expensive) check C ⊆ C'
```

Variable elimination

```
tryToEliminate (Variable x)
    // tries to eliminate variable x
    if x has zero occurrences then return
    if occurList(x) > 10 and occurList(x̄) > 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 vas 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:

- Niklas Eén, Armin Biere. *Effective Preprocessing in SAT Through Variable and Clause Elimination.* SAT 2005: 61-75

Other resources:

- `http://minisat.se/SatELite.html`