MiniMarch

Siert Wieringa *s.wieringa@student.tudelft.nl*

Supervisors:
H. van Maaren *h.vanmaaren@tudelft.nl*
M. Heule *marijn@heule.nl*

January 30th, 2007

MiniMarch is a modified version of MiniSat 2.0. The design goal was to make a version of MiniSat that is less sensitive to shuffling of the input formula. MiniMarch extends MiniSat with the following techniques:

- **Clause sorting** By sorting the clauses we not only hope to find conflict clauses imposing strong constraints faster but also facilitate resistance against the shuffling of clauses in the input formula.
- **Symmetric simplifier** The simplifying pre-processor introduced in MiniSat 2.0 is rather sensitive to the swapping of signs in the input formula. Our modifications make it symmetric.
- **Branching heuristics** MiniMarch combines dynamic lookahead branching heuristics for balanced variables with static occurrence based heuristics for unbalanced variables.
- **Activity initialization** The activities are initialized, to make the initial variable decisions taken less sensitive to variable index shuffling.

1. Clause sorting

Clauses are sorted, the first sorting criterion is the shortest size first. Amongst clauses of the same length the clauses with the highest weight w_{clause} are ordered first.

$$w_{clause}(c) = \sum_{l \in c} w_{literal}(l)$$

$$w_{literal}(l) = \sum_{c \in Clauses \ \wedge \ l \in c} \frac{1}{2^{|c|-1}}$$

2. Symmetric simplifier

The simplifier in MiniSat 2.0 attempts to eliminate variables from the formula. In an attempt to eliminate variable v the solver does a *merge* on all pairs of clauses with literal v occurring in one and literal $\neg v$ in the other. The result of a *merge* is the clause that consists of all literals in both clauses minus v and $\neg v$.

The simplifier used to call *merge* on the clauses from the list using the scheme described in algorithm 1.

Algorithm 1 Clause merging old style

```
1: P := Set of all clauses containing literal v
2: N := Set of all clauses containing literal ¬v
3: for all clauses p in P do
4:    for all clauses n in N do
5:       merge( p, n )
6:    end for
7: end for
```

To make the simplifier unaffected by shuffling we have modified this scheme. We require the subset of clauses in which v occurs to be sorted as described in section 1. The new scheme is now based on choosing pairs of clauses that are best in the ordering first.

Algorithm 2 Clause merging new style

```
1: C_v := Sorted set of all clauses containing v
2: for i=0 to |C_v| − 1 do
3:    for j=i + 1 to |C_v| do
4:       if polarity of v in C_v[i] ≠
               polarity of v in C_v[j] then
5:          merge( C_v[i], C_v[j] )
6:       end if
7:    end for
8: end for
```

3. Branching heuristics

MiniMarch combines dynamic lookahead branching heuristics for balanced variables with static occurrence based heuristics for unbalanced variables. The balance of a variable is defined as:

$$bal(v) = \frac{w_{literal}(v) + 1}{w_{literal}(\neg v) + 1} + \frac{w_{literal}(\neg v) + 1}{w_{literal}(v) + 1} - 2$$

A variable is regarded as balanced if it's balance is smaller then the balance of the whole formula which is defined by:

$$w_{poslit} = \sum_{l \in Variables} w_{literal}(l)$$

$$w_{neglit} = \sum_{\neg l \in Variables} w_{literal}(\neg l)$$

$$balance = \frac{w_{poslit} + 1}{w_{neglit} + 1} + \frac{w_{neglit} + 1}{w_{poslit} + 1} - 2$$

For unbalanced variables ($bal(v) > balance$) the branching direction $d(v)$ is determined as the direction of the literal with the lowest literal weight, $w_{literal}$. For balanced variables this same value $d(v)$ is determined as a first estimate to the best branching direction. It might be undefined if $w_{literal}(\neg v) = w_{literal}(v)$.

Whenever the activity based branching heuristic used by MiniSat picks a balanced variable the lookahead procedure described in algorithm 3 is invoked.

Algorithm 3 LOOKAHEAD(l)

```
1:  oldLevel := backtrackLevel
2:  propQ1 := PROPAGATE(l)
3:  if propQ1 = conflict then
4:    return
5:  else
6:    BACKTRACK(oldLevel)
7:    propQ2 := PROPAGATE(¬l)
8:    if propQ2 ≠ conflict and
          propQ1 > propQ2 then
9:      BACKTRACK(oldLevel)
10:     PROPAGATE(l)
11:   end if
12: end if
```

The literal l that the lookahead procedure starts with is the opposite of the literal determined by $d(v)$. This is because lookahead will have to propagate both literals and decide on the best. If the literal propagated second has the highest chance of being the best the chance of having to revert to the literal propagated first goes down. If $d(v)$ has not yet been defined it will be defined as the literal with the highest number of clause watches or if that is equal the negation of the literal of v that occurred first in the sorted set of clauses. The variables $propQ_1$ and $propQ_2$ return a measurement indicating the "quality of the propagation". They can be either a natural number or $conflict$. If no conflict was found the value is defined as:

$$propQ(v) = |IUP(v)| * \sum_{r \in IUP(v)} m(r)$$

Where IUP(v) is the set of all variables of which the value became fixed as a result of iterative unit propagation resulting of the propagation of v (including v itself). The value of $m(r)$ is the sum of the distances the watch pointers moved to the next undefined variable in a clause that was watching r.

4. Activity initialization

The MiniMarch solver uses activity initialization, a technique that was proposed before by R.H. Kibria (2006). The actual value of the initial activity in MiniMarch differs from the one proposed in that earlier work, we use:

$$activity_{initial}(v) = \frac{\#clauses - i_v}{\#clauses}$$

With i_v the index of the clause in which v occurs for the first time in the sorted set of clauses.

5. Acknowledgement

Special thanks to INOXA internet (www.inoxa.nl) for supporting this project by making a brand new server available for the purpose of testing MiniMarch.

References

[1] R.H. Kibria (2006), Actin solver description submitted to SAT Race 2006

[2] MiniSat v2.0 Beta, Niklas Eén, Niklas Sörensson