
```

Procedure BTOSat( $S$ )
 $S := PropUnit(S)$ ;
while  $\square \notin S$  and  $S \neq \emptyset$  do:
  choose an unassigned literal  $x$ ;
   $R := PropUnit(S \cup \{x\})$ ;
  if  $\square \in R$ 
  then  $S := PropUnit(S \cup \{\bar{x}\})$ 
  else  $S := R$ ;
endwhile
If  $\square \in S$ 
then return “Unsatisfiable”
else return the current assignment;

```

Figure 1: Backtrack-once algorithm

The “standard” 2-SAT algorithm

The classical “guess and deduce” algorithm for 2-SAT was introduced in (Even, Itai, & Shamir 1976); we provide here a modernized version of the algorithm, similar to that provided in (Dalal & Etherington 1992). The algorithm, which we will call *BTOSat* (for “backtrack once”), is described in Figure 1. It can be seen as a restricted case of the classic Davis-Putnam backtracking-plus-unit-resolution procedure for general SAT problems (Davis, Logemann, & Loveland 1962). *PropUnit* stands for the well-known algorithm for unit resolution used in almost all SAT solvers, also called unit propagation or boolean constraint propagation, which runs in time $O(|S|)$. Good descriptions of *PropUnit* can be found in e.g. (Dowling & Gallier 1984; Dalal & Etherington 1992; Forbus & de Kleer 1993). *BTOSat* simply chooses literals to assign, and propagates their value with unit resolution. If an assignment x yields a contradiction, then it adds \bar{x} to the input S ; otherwise it adds x to S . The crucial aspect is that *BTOSat* never backtracks to previously assigned variables. We may need to undo an assignment to x , but if the complementary assignment \bar{x} ends in failure then the theory is unsatisfiable. Hence *BTOSat* backtracks at most one step.

As described (but see discussion of the “parallel version” later), *BTOSat* has complexity $O(nm)$. The following two examples show that this bound is tight, illustrating two different sources of redundancy in *BTOSat*: respectively, the failure to identify the source of contradictions, and the repeated propagation of values from branches which lead to contradictions, even when those values are not responsible for the contradiction.

Example 1 Consider the theory $S_1 = \{\bar{x}_1x_2, \bar{x}_2x_3, \dots, \bar{x}_{n-1}x_n, \bar{x}_{n-1}\bar{x}_n, x_{n-1}\bar{x}_n, x_{n-1}x_n\}$. Note that the last four clauses are unsatisfiable. Suppose literals are assigned in their natural order, preferring always the positive literal. This leads to a sequence of $n - 1$ failures, with $PropUnit(S_1 \cup \{x_1\})$, $PropUnit(S_1 \cup \{\bar{x}_1, x_2\})$, and so on, returning \square . In each step, the procedure must consider all remaining unsubsumed clauses. It is only in the last step that S_1 is determined unsatisfiable when both $PropUnit(S_1 \cup \{\bar{x}_1, \dots, \bar{x}_{n-2}x_{n-1}\})$ and $PropUnit(S_1 \cup \{\bar{x}_1, \dots, \bar{x}_{n-1}\})$ yield \square . \square

```

Procedure TempPropUnit( $x$ )
/* Input: A literal  $x$  to be tentatively assigned. */

if  $tempval(x) = false$  /*temporary conflict,  $S \models \neg x \supset x$  */
then set  $S := PropUnit(S \cup \{x\})$  and return;
 $tempval(x) := true$ ;  $tempval(\bar{x}) := false$ ;
foreach  $y\bar{x} \in S$  do:
  if  $\square \in S$  then return;
  if  $tempval(y) \neq true$  then TempPropUnit( $y$ );

```

```

Procedure BinSat( $S$ )
/* Input: A binary clausal theory  $S$  */

```

```

foreach variable  $p$  of  $S$  do:
   $tempval(p) := tempval(\bar{p}) := NIL$ ;
   $permval(p) := permval(\bar{p}) := NIL$ ;
 $S := PropUnit(S)$ ;
while ( $\square \notin S$  and there exists a literal  $x$ 
  s.t.  $permval(x) = tempval(x) = NIL$ ) do:
  TempPropUnit( $x$ );
If  $\square \in S$ 
then return Unsatisfiable;
else return Satisfiable;

```

Figure 2: Linear time algorithm for 2-SAT. *PropUnit* assigns variables by setting *permval*’s (see text).

In this example, we can solve the problem if we detect in the first step that the source of the contradiction in each of the failed assignments x_i is that $S_1 \models \bar{x}_{n-1}$, from which the unsatisfiability of S_1 can be immediately detected by adding \bar{x}_{n-1} to S_1 and running *PropUnit*.

Detecting the source of contradictions is not enough, however, to avoid $\Theta(nm)$ cost:

Example 2 Consider the following theory $S_2 = \{\bar{x}_1x_2, \bar{x}_2x_3, \dots, \bar{x}_{k-1}x_k, \bar{x}_ky_1, \bar{y}_1y_2, \dots, \bar{y}_{k-1}y_k, \bar{x}_1z, \bar{x}_1\bar{z}, \dots, \bar{x}_kz, \bar{x}_k\bar{z}\}$. Suppose again we assign first x_1, \dots, x_n , positively. Assigning any of the x_i ’s positively yields a contradiction using the second row of clauses. Before doing so, however, it may assign all later x_j ’s as well as all the y_i ’s. While replacing the assignment, say, x_1 , by \bar{x}_1 does correctly identify the source of contradiction, it also forces us to undo all other assignments to the y_i ’s. This is unfortunate, as these are perfectly fine, and *BTOSat* will in fact reassign them as a by-product of each of the x_i assignments. \square

These two examples are sufficient to motivate the new algorithm, described in the next section.

A new linear time algorithm for 2-SAT

BinSat, the new algorithm for 2-SAT, is described in Figure 2. *BinSat* simply tries assignments such as *BTOSat*, using the routine *TempPropUnit*, abbreviated *TPU*. *TPU*(x) takes a literal x to be temporarily assigned and propagates its value by unit resolution. It works in a depth first fashion, assigning temporary values *tempval* to x and to every literal derivable by unit resolution from $S \cup \{x\}$. When it is about to assign a *tempval* to a literal y which is in contradiction with the previous *tempval*(y), it recognizes this as the entailment

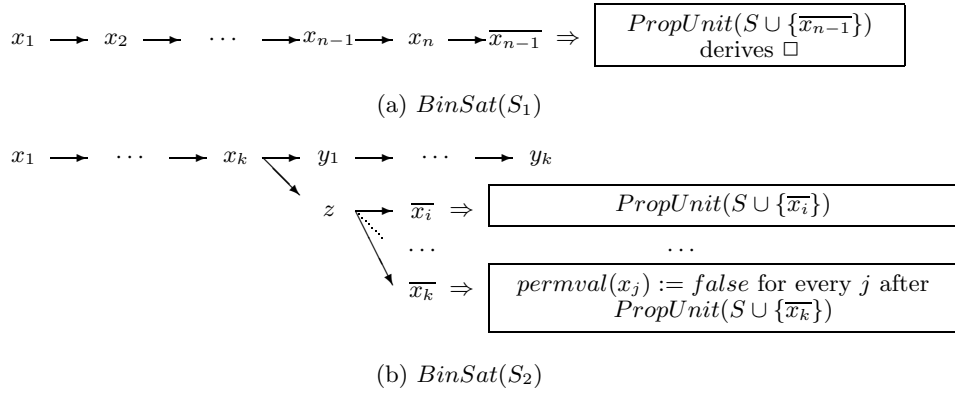


Figure 3: Examples of *BinSat*. An arrow $x_i \rightarrow x_j$ denotes that $TPU(x_j)$ is called from the loop of $TPU(x_i)$.

$S \models \bar{y} \supset y$, i.e. $S \models y$, and, through a call to *PropUnit*, it *permanently* assigns the value true to y and propagates it. We represent the permanent character of the effects of *PropUnit* by assuming that, for any x which it propagates, it sets $permval(x) := true$, $permval(\bar{x}) = false$, and effectively removes from S clauses containing x (in practice, it may simply mark these clauses as permanently subsumed). Temporary assignments are ignored by *PropUnit*.

BinSat does not revoke any assignment unless forced to; correctly identifies the source of conflict; and *does not stop unit propagation when a contradiction is found*—instead, it searches for, and finds, all conflicts, whether they are induced by the initial literal which was tentatively assigned, or by a literal that was assigned as a result of the initial tentative assignment. As a result, any tentative assignment is explored in full only once, and revoked at most once.

If *BinSat* returns Satisfiable, a satisfying assignment can be found as follows: for each variable p , if $permval(p) \neq NIL$ then p is assigned $permval(p)$; otherwise, p is assigned $tempval(p)$.

Example 3 The execution of *BinSat* on S_1 , with the same ordering of literals, is depicted in Figure 3.a. *BinSat*(S_1) recursively calls in sequence $TPU(x_1), \dots, TPU(x_{n-1}), TPU(x_n)$, and finally $TPU(\bar{x}_{n-1})$. At this point, x_{n-1} cannot be assigned, and the algorithm calls $S := PropUnit(S_1 \cup \{\bar{x}_{n-1}\})$, which derives a contradiction from the last two clauses.

Note that backjumping, which is usually described as “identifying the source of conflict,” cannot help here; indeed the strength of backjumping is in “jumping back” more than one step at a time, which is never needed for 2-SAT. Clearly, certain conflicts are not identified by backjumping, which would only add useless overhead for these instances. \square

Example 4 As mentioned, in order to avoid repeating any work, *BinSat* goes beyond correctly identifying culprits of contradictions. Figure 3.b illustrates a possible runs of *BinSat*(S_2), with the same ordering of literals as in example 2. Note that multiple conflicts can be discovered within a single top level *TPU* call, as can be

seen in the various calls to *PropUnit*, and that the y_i assignments are never undone. \square

Lemma 1 *Suppose $tempval(x) = false$ right before some call to $TPU(x)$. Then $S \models x$ (and therefore S is satisfiable iff $PropUnit(S \cup \{x\})$ is satisfiable).*

Lemma 2 *Suppose $tempval(x) := true$ is assigned in a $TPU(x)$ call. Then $tempval(x)$ never changes, and $TPU(x)$ is never called again.*

By lemma 1 and soundness of *PropUnit*, all permanent assignments made by *BinSat* preserve satisfiability. By lemma 2, *TPU* is called at most once per literal, and thus no clause is considered more than twice within *TPU* calls. Similar analysis can be applied to *PropUnit* and *BinSat* as a whole. We can show:

Theorem 3 *BinSat correctly decides the satisfiability of binary clausal theories, in time $O(m)$.*

This analysis assumes that for an $O(m)$ initialization step by which clauses are indexed by the literals they contain, so that the loop of $TPU(x)$ considers exactly those y 's such that $\bar{x}y \in S$.

Interestingly, tentative assignments which are not revoked by an opposite permanent value by the end of the toplevel call in which they were made, are never revoked. Thus they behave exactly as permanent assignments.¹ See e.g. the y_i 's in Figure 3.b.

Related work on 2-SAT

As said, *BTOSat* originates in (Even, Itai, & Shamir 1976). As first suggested by Even et al., *BTOSat* can be implemented in parallel $O(m)$ time by working in parallel in the branches for p and \bar{p} ; the first branch that finishes without returning a contradiction gets its assignment made permanent, immediately interrupting work on the other branch. If both branches return a contradiction, then the theory is unsatisfiable. We can obtain

¹Note however that an unrevoked tentative value can be *confirmed* by an identical permanent value. This source of redundancy in *BinSat* has an easy fix: before assigning some $permval(x)$ in *PropUnit*, check that that $tempval(x)$ is not already set to the same value; if it is, do not add x to the stack of unit clauses to be propagated.

the same complexity with a single processor, by interleaving work in alternative branches. But this is obviously more complex to implement than *BinSat*, and may lead to twice more work than needed, as we always explore two branches.

Define the *implication graph* $G(S)$ as the graph whose nodes are the literals of S , and there is an edge from a literal v to a literal w iff there exists a clause $\bar{v}w \in S$. (Aspvall, Plass, & Tarjan 1979) provides a linear algorithm based on the $G(S)$ graph. The algorithm first finds the strongly connected components of $G(S)$; members of the same SCC are literals which must have the same truth value in any satisfying assignment. Then it generates an assignment by processing the SCCs in topological order. Finding SCCs is usually done by two complete depth-first searches of the graph, inverting all edges before the second pass; a satisfying assignment is generated by outputting the SCCs in topological order. *BinSat* can be described as performing a single *partial* depth-first search of this graph: a call to $TPU(y)$ from $TPU(x)$ traverses the (x, y) edge corresponding to the clause $\bar{x}y$, in which case the (\bar{y}, \bar{x}) edge is traversed only if there's a call to *PropUnit* which inverts the temporary assignments. This partial search will often be shortcircuited by derivation of permanent values, and generates the assignment on the fly. Thus *BinSat* should be more efficient.²

After developing *BinSat*, we learned about the linear 2-SAT algorithm of (Chandru *et al.* 1990), a special case of an algorithm for renamable Horn recognition. One can find some similarity with the ideas of *BinSat* after some digging, by mapping their graph traversal procedures to implicit unit resolution operations. Their algorithm relies on a substantially more complicated graph than $G(S)$, having both variables and clauses as vertices. In addition, a referee pointed out that (Pretolani 1993) provides a hypergraph version of the algorithm of (Chandru *et al.* 1990). We defer discussion of these algorithms to the section on related work in renamable Horn.

Horn renamability

Recall that a theory is renamable Horn iff there exists a uniform renaming of its variables such that the theory becomes Horn after the renaming. For example, the theory $\{ab, cd\}$ can be made Horn by renaming a to be the negative literal \bar{a}^* , and c to be \bar{c}^* , where both a^* and c^* are new variables. Equivalently, S is renamable Horn iff there exists an interpretation such that at most one literal per clause is false in this interpretation. As it is well known, unit resolution is a complete satisfiability method for renamable Horn, not just for Horn. Horn renamability can be reduced to the satisfiability of a set of binary clauses. Specifically, for any set of clauses S , let S_B be the binary theory consisting of all clauses xy such that the literals x and y occur jointly in some clause of S . Then S is renamable Horn iff S_B is satisfiable (Lewis 1978). Generating S_B requires quadratic

²Note that Aspvall's method finds *all* pairs of literals such that $S \models l_1 \equiv l_2$ (when l_1 and l_2 are in the same SCC) in order to detect whether $S \models x \equiv \bar{x}$ for some x .

Procedure RH-Prop(x)

/ Input: A literal x to be permanently assigned in S_B . */*

```

permval( $x$ ) := true; permval( $\bar{x}$ ) := false;
 $Q := \{x\}$ ; /* stack of literals whose permval is true */
while  $Q \neq \emptyset$  do:
   $y := pop(Q)$ ;
  foreach clause  $C \in S$  s.t.  $\bar{y} \in C$  do:
    foreach literal  $z \in C \setminus \{\bar{y}\}$  do:
      if permval( $z$ ) = false
        then /*  $S_B$  unsatisfiable */
           renamable := false; return;
      else if permval( $z$ ) = NIL
        then {permval( $z$ ) := true; permval( $\bar{z}$ ) := false;
             push( $z, Q$ );}

```

Procedure RH-TempProp(x)

/ Input: A literal x to be tentatively assigned in S_B */*

```

if tempval( $x$ ) = false /*  $S_B \models \neg x \supset x$  */
  then return RH-Prop( $x$ );
tempval( $x$ ) := true; tempval( $\bar{x}$ ) := false;
foreach clause  $C \in S$  s.t.  $\bar{x} \in C$  do:
  foreach literal  $z \in C \setminus \{\bar{x}\}$  do:
    if renamable = false then return;
    if tempval( $z$ )  $\neq$  true and permval( $z$ )  $\neq$  true
      then RH-TempProp( $z$ );

```

Figure 4: Auxiliary routines for renamable Horn.

time and space, respectively $O(mn^2)$ and $O(n^2)$. While the cost of generating a binary theory corresponding to the Horn renamability problem can be reduced to $O(|S|)$ (see the discussion of (Aspvall 1980) later), we can in fact skip this step altogether, as we show next.

Figures 4 and 5 describe a new $O(|S|)$ algorithm, *RHSat*(S), which decides both membership in the class renamable Horn, and satisfiability of renamable Horn and binary theories. It works in a similar manner to *BinSat*, except that no binary theory is explicitly constructed, and no new variables need to be reasoned with. As a result, *RHSat* is likely to be much more efficient in practice than competing algorithms.

The easiest way to understand *RHSat*, procedurally, is as the repeated application of the rule (RH) “from a clause $xC \in S$ and literal \bar{x} to be propagated, assign *all* literals of C as *true*.” This rule is applied in a two-level fashion, just as in *BinSat*, corresponding to tentative and permanent assignments, through the subroutines *RH-TempProp* and *RH-Prop*, respectively. The main loop of *RHSat* simply handles literals to try and assign to *RH-TempProp*.

We come back to this rule later. We will instead describe here the procedure in terms of a simulation of unit resolution over an implicit S_B . *RHSat* mimics *BinSat* in the relevant parts, but the subroutines are adapted to work on this implicit S_B . The key observation is that unit propagation of values in S_B can be done directly by considering the clauses of S . Given a clause $x_1x_2 \dots x_k \in S$, a call $TPU(\bar{x}_1)$ in S_B would call TPU for each of x_2, \dots, x_k ; and similarly for propagating \bar{x}_1

Procedure $RHSat(S)$

/ Input: A set of clauses S . Output: See theorem 4. */*

foreach variable p of S **do**:
 $tempval(p) := tempval(\bar{p}) := NIL$;
 $permval(p) := permval(\bar{p}) := NIL$;
 $S := PropUnit(S)$;
if $\square \in S$ **then return** Unsatisfiable;
 $renamable := true$;
while ($renamable = true$ **and** there exists a literal x
 s.t. $permval(x) = tempval(x) = NIL$) **do**:
 $RH-TempProp(x)$;
if $renamable = false$
then {**if** S is binary
 then return Unsatisfiable;
 else return Non-renamable; }
else return Satisfiable;

Figure 5: Algorithm for renamable Horn theories.

with $PropUnit$ over S_B . There is therefore no need to generate S_B explicitly in order to run unit propagation over S_B .

The main procedure $RHSat$ first sets $S := PropUnit(S)$, and thereafter works on the simplified problem without unit clauses. Let us keep using S_B for the binary theory encoding the renamability problem for $PropUnit(S)$. Then $RHSat$ iterates over unassigned literals just as $BinSat$, working over the implicit S_B by tentatively assigning literals with $RH-TempProp$. $RH-TempProp$ corresponds to TPU over S_B , exploring tentative assignments in full, and $RH-Prop$ corresponds roughly to $PropUnit$, setting and propagating permanent assignments over S_B which have been derived by $RH-TempProp$. Note that in all routines, the termination test $\square \in S$ which was used in $BinSat$ is replaced by the test $renamable = false$, since the fact that $S_B \models \square$ means only that S is not renamable, not that it is unsatisfiable. Unless, of course, $PropUnit(S)$ is binary, in which case $PropUnit(S) = S_B$, and S is satisfiable iff $PropUnit(S)$ is renamable Horn.

Example 5 Let $S_4 = \{\bar{q}p, \bar{r}p, \bar{p}st, \bar{s}q, \bar{t}r\}$, and let $S_5 = S_4 \cup \{\bar{q}\bar{r}\}$. The former is renamable Horn, while the latter is not. Figure 6 provides possible runs of $RHSat$ over each of these inputs. We illustrate two potential runs with S_4 , both of which yield the assignment p, q, r, s, t ; this can be interpreted as requiring that the sign of all variables is changed (see discussion below). In fact, these changes are all forced, as illustrated in the second run; that is, renaming all variables is the *only* way to make S_4 Horn. In the second example, with S_5 , the call $RH-TempProp(\bar{p})$ results in a call to $RH-Prop(\bar{p})$, which in turn ends up in failure.

There is a very natural reading of the figures in terms of renaming. A positive literal is a variable saying “rename me,” a negative literal a variable saying “do not rename me.” For example, Figure 6.c “reads” as follows: if you change the sign of p then you must change the sign of s as well (otherwise both p and s would both be positive in the renamed $\bar{p}st$, which therefore would not be Horn); but

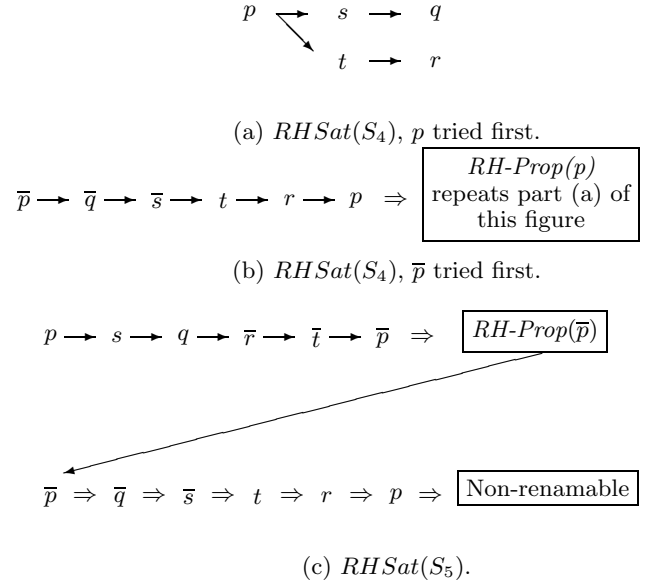


Figure 6: Examples of $RHSat$. An arrow $x \rightarrow y$ denotes that $RH-TempProp(y)$ is called from the loop of $RH-TempProp(x)$. Similarly, $x \Rightarrow y$ means that $RH-Prop$ derives y from x .

changing s requires renaming q , because of $\bar{s}q$; renaming q in turn forces you to keep r unchanged, etc. \square

Theorem 4 *If $\square \in PropUnit(S)$, or if $PropUnit(S)$ is renamable Horn or binary, then $RHSat(S)$ correctly decides the satisfiability of S . Otherwise, $RHSat(S)$ correctly returns Non-renamable. The running time is $O(|S|)$.*

Corollary 5 *$RHSat$ is a satisfiability decision procedure exactly for the class of theories S such that $PropUnit(S)$ is renamable Horn or binary, or S contains an unsatisfiable subset which is renamable Horn.*

Note that $PropUnit(S)$ may be renamable Horn or binary even if S is neither, and thus that $RHSat$ decides a larger class than the union of renamable Horn and binary, with a single algorithm. The reasoning for the $O(|S|)$ bound is similar to that for $BinSat$. We can show that $RH-Prop$ and $RH-TempProp$ can each consider a clause at most twice. We use the same indexing scheme as in $BinSat$, a list of clauses in which each literal occurs. This scheme is used by most SAT solvers; our algorithms require no additional data structures beyond these.

We have focused on satisfiability as opposed to recognition per se. It is trivial to modify $RHSat$ to make it a pure recognition algorithm, if desired. If the theory is determined satisfiable, then a satisfying assignment can be found exactly as in $BinSat$ (i.e. use the $permval$'s if available, otherwise the $tempval$'s). This assignment satisfies all but possibly one literal from each clause of S , since it satisfies S_B . The renaming to make S Horn can be obtained from this assignment: if a variable is assigned true, then change its sign in S .

As we have mentioned, the renamable Horn algorithm

can be understood simply as repeated application of the rule (RH) given above. Both *RH-Prop* and *RH-TempProp* can be seen as applying this rule, respectively with permanent and tentative assignments. Consider the semantics by which an assignment is said to satisfy a non-unit clause iff it satisfies all but possibly one of its literals. It turns out that a theory is renamable Horn iff there is some assignment that satisfies *in this sense* all non-unit clauses of S . (If S is Horn, this assignment is the one which assigns *false* to all variables.) Clearly, (RH) is sound with respect to this semantics, and can be made complete using the two level-approach of *RHSat*, as shown by Theorem 4. Hence the propagation rule can be seen as either mimicing unit propagation over S_B without generating it; or simply as a direct implementation of the semantics just described as a means to identify renamable Horn.

Related work on renamable Horn

Alternative algorithms for recognizing renamable Horn theories can be found in (Chandru *et al.* 1990; Pretolani 1993; Aspvall 1980). We have already mentioned the first two algorithms in the context of 2-SAT; again, the algorithm of (Pretolani 1993), which was pointed to us by a referee and which we haven't been able to examine in detail in the short time allotted for revision of this paper, is described as an hypergraph version of (Chandru *et al.* 1990). The correspondence between these two algorithms appears to be as follows: a clause node in (Chandru *et al.* 1990) corresponds in Pretolani's algorithm to a directed hyperedge from variables occurring negatively (or from *false* if none) to variables occurring positively in the clause (or to *true* if none). *BinSat* and *RHSat* are significantly simpler than either algorithm, both in terms of data structures and, equally importantly, of the conceptual baggage needed to understand them.

A different approach was followed earlier by Aspvall (Aspvall 1980). He showed that by introducing auxiliary variables it is possible to generate in $O(|S|)$ time and space a binary theory S_B^* which is satisfiable iff S_B is satisfiable. Coupled with *BinSat* or any other linear 2-SAT algorithm, Horn renamability can be decided in linear time by first generating S_B^* and then testing it for satisfiability.

Since we do not need to generate any binary theory, our algorithm is more efficient than the procedure just described. Furthermore, it can be shown that $|S_B^*| = (6 - 8/k)|S|$, assuming all clauses of S have length $k > 2$. Multiplying our memory requirements almost sixfold will not always be an option. Even if it is, simply "preparing the input" may take six times more time than with *RHSat* (which basically has only to read S , as opposed to generating S_B^*); and "actually solving the problem" with *RHSat* needs only consider $|S|$ literal occurrences, as opposed to, again, a sixfold increase. The generation cost, furthermore, is always paid by Aspvall's method, even when it may happen that the resulting binary theory is easily unsatisfiable (i.e. *RHSat* may be able to detect unsatisfiability by considering only a small subset of clauses of S).

Conclusion

We introduce new linear time algorithms for 2-SAT and renamable Horn, which improve on previous algorithms in efficiency and simplicity. Being based on unit resolution, they are likely to be more easily integrated into general SAT solvers based on the standard scheme of backtracking plus unit resolution. We provide one explicit example in the extended version of the paper by showing how to optimize the enumeration of models of a binary theory, which could be used by a SAT solver such as Nemesis (Larrabee 1992), with complexity almost linear per model. Other applications are possible, for example in the hierarchy of tractable classes $\{R_i\}$, defined by (Pretolani 1996), where R_0 is renamable Horn, and $S \in R_i$ iff either $S \in R_{i-1}$ or there exists a literal x such that $PropUnit(S \cup \{x\}) \in R_{i-1}$ and $PropUnit(S \cup \{\bar{x}\}) \in R_i$.

References

- Aspvall, B.; Plass, M. F.; and Tarjan, R. E. 1979. A linear-time algorithm for testing the truth of certain quantified Boolean formulas. *Information Processing Letters* 8(3):121–123.
- Aspvall, B. 1980. Recognizing disguised NR(1) instances of the satisfiability problem. *Journal of Algorithms* 1(1):97–103. Note.
- Chandru, V.; Coullard, C.; Hammer, P.; Montañez, M.; and Sun, X. 1990. On renamable Horn and generalized Horn functions. *Annals of Mathematics and Artificial Intelligence* 1:33–48.
- Dalal, M., and Etherington, D. W. 1992. A hierarchy of tractable satisfiability problems. *Information Processing Letters* 44:173–180.
- Davis, M.; Logemann, G.; and Loveland, D. 1962. A machine program for theorem proving. *Communications of the ACM* 5:394–397.
- Dowling, W. F., and Gallier, J. H. 1984. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming* 3:267–284.
- Even, S.; Itai, A.; and Shamir, A. 1976. On the complexity of timetable and multicommodity flow problems. *SIAM Journal of Computing* 5(4):691–703.
- Forbus, K., and de Kleer, J. 1993. *Building Problem Solvers*. The MIT Press.
- Henschen, L., and Wos, L. 1974. Unit refutations and Horn sets. *Journal of the ACM* 21:590–605.
- Larrabee, T. 1992. Test pattern generation using boolean satisfiability. *IEEE Transactions on Computer-Aided Design* 4–15.
- Lewis, H. R. 1978. Renaming a set of clauses as a Horn set. *Journal of the ACM* 25(1):134–135.
- Pretolani, D. 1993. *Satisfiability and Hypergraphs*. Ph.D. Dissertation, Università di Pisa.
- Pretolani, D. 1996. Hierarchies of polynomially solvable satisfiability problems. *Annals of Mathematics and Artificial Intelligence* 17:339–357.