Automated Theorem Proving *– Foundations of SAT-Solving –*

July 2015

Wolfgang Küchlin

Symbolic Computation Group
Wilhelm-Schickard-Institute of Informatics
Faculty of Mathematics and Sciences

Universität Tübingen

Steinbeis Transferzentrum
Objekt- und Internet-Technologien (OIT)

Wolfgang.Kuechlin@uni-tuebingen.de
http://www-sr.informatik.uni-tuebingen.de

SR

Contents

- Propositional Resolution
  - Theorem Proving by Deduction
  - From 2 clauses C_1 and C_2, deduce new resolvent clause R where $C_1 \wedge C_2 \vDash R$, hence $\{C_1, C_2\} \equiv \{C_1, C_2, R\}$
  - Prove SAT($\mathcal{F}$)=false by deducing $\bot$ (represented by unsatis-fiable *empty clause* { } =: □) since $\mathcal{F} \equiv \mathcal{F} \cup \{\,\} \equiv \mathcal{F} \wedge \bot \equiv \bot$
- Elementary DPLL SAT-Solving
  - The historical Davis-Putnam-Logemann-Loveland Method
  - Compute SAT(F) *in place* by
    - Trying some variable assignment x=true
    - Computing other variable assignments as immediate consequences
    - Carrying on, until success, or backtracking to x=false.

The Resolution Rule

- (sole) inference rule:

$$\frac{C \cup \{\ell\} \qquad D \cup \{\neg\ell\}}{C \cup D}\,\text{Res}$$

  where C´ = C ∪ {ℓ} and D´ = D ∪ {¬ℓ} are clauses.

  The *resolvent* R = C ∪ D is deduced by *resolving parent clauses* C´ and D´ *on the literal ℓ.* We write {C´, D´} ⊢ R.

- Examples:

$$\frac{\{x, y, \neg z\} \quad \{u, \neg v, z\}}{\{x, y, u, \neg v\}} \qquad \frac{\{x, u, \neg v\} \quad \{\neg u\}}{\{x, \neg v\}}$$

Example: Resolution proof-tree

F3={{x,y},{x,¬y,z},{¬x},{¬y,¬z}}

$$\begin{tikzcd}
 & & & \{x,y\} \arrow[dr, no head] & & \{x,\neg y,z\} \arrow[dl, no head] & \\
 & & & & \{x,z\} \arrow[d, no head] & \{\neg x\} \arrow[dl, no head] & \\
\{x,y\} \arrow[dr, no head] & & \{\neg x\} \arrow[dl, no head] & & \{z\} \arrow[dr, no head] & & \{\neg y,\neg z\} \arrow[dl, no head] \\
 & \{y\} \arrow[drr, no head] & & & & \{\neg y\} \arrow[dll, no head] & \\
 & & & \square & & &
\end{tikzcd}$$

Clauses and Implications, Soundness, Completeness

- A clause embeds a variety of implications:
  - Example: $(x \vee y \vee z) \equiv \neg(\neg x \wedge \neg y) \vee z \equiv (\neg x \wedge \neg y \rightarrow z)$
  - But also: $(x \vee y \vee z) \equiv \neg(\neg x) \vee (y \vee z) \equiv (\neg x \rightarrow y \vee z)$
- Resolution represents deduction of implications
  - Parent clauses $(A \vee x)$ and $(B \vee \neg x)$, resolvent $(A \vee B)$
  - $(A \vee x) \equiv (\neg(\neg A) \vee x) \equiv (\neg A \rightarrow x)$
  - $(B \vee \neg x) \equiv (x \rightarrow B)$
  - From $(\neg A \rightarrow x)$ and $(x \rightarrow B)$ follows $(\neg A \rightarrow B) \equiv (\neg(\neg A) \vee B) \equiv (A \vee B)$
- The resolution calculus is *sound* (sound). For all sets $\mathcal{F}$ of clauses and all clauses D, we have: $\mathcal{F} \vdash^*_{Res} D$ implies $\mathcal{F} \vDash D$
- Hence: $\mathcal{F} \vdash^*_{Res} D$ implies $\mathcal{F} \equiv \mathcal{F} \cup D$
- Theorem: the resolution calculus is *refutation complete:*

  $\mathcal{F} \vDash \bot$ implies $\mathcal{F} \vdash^*_{Res} \square$

Subsumption and Unit Resolution

- Clause C *subsumes* clause D iff $C \subseteq D$.
  - Constraint D is subsumed by C, because C is stronger.
- If $C \subseteq D$ then $C \vDash D$.
  - Because $D \equiv C \vee R$ for some R
- Lemma: Subsumed clauses can be cancelled
  - Let $C \subseteq D$. Then $\mathcal{F} \cup C \equiv \mathcal{F} \cup C \cup D$.
  - Absorption Law: $F \wedge (F \vee G) \equiv F$
- In *Unit Resolution* one of the parent clauses is a *Unit* (singleton)
  - For parent clauses $(A \vee x)$ and $(\neg x)$, the resolvent is (A)
  - Unit resolution is valuable, because A is shorter than $(A \vee x)$.
  - Moreover, the (shorter) resolvent subsumes the longer parent: $A \wedge (A \vee x) \equiv A$
  - Hence the parent can be cancelled

Resolution proof procedure

- Let C be a set of axioms (constraints). In order to prove a theorem D, $C \vDash D$, proceed as follows:

1. Negate D. Convert $C \cup \neg D$ into CNF. Call the result F.
2. Repeat
   - i. Compute all non-tautological resolvents R from F
   - ii. If $\square \in R$, return „proof“.
   - iii. Delete from R all subsumed clauses. // forward subsumption
   - iv. If R = { }, return „disproof“. // $\square$ not found, no more deductions possible
   - v. Delete from F all clauses subsumed by R. // backward subsumption
   - vi. $F := F \cup R$

- Theorem [refutation completeness]: If $C \vDash D$, the proc. stops in (ii)
- Otherwise the procedure stops in (iv), because there are at most 3^n subsumption-free clauses in n variables (x, ¬x, don´t care x).
- Bad news: far too many useless deductions, memory explodes.

SAT-Solving with the DPLL Algorithm

- DPLL: Davis-Putnam-Logemann-Loveland
  - DPLL dates from 1960/62, significantly improved to CDCL
  - CDCL (conflict driven clause learning) ~1996
- Decision procedure for *SAT-Solving* problem SAT(F)
  - Compute SAT(F)=? Is there a satisfying assignment for *F*?
  - Alternative formulation: eliminate quantifiers in $\exists x_1,\ldots,x_n$:F
  - Idea: Iteratively try to construct an assignment, propagating the consequences at each step, backtrack exhaustively if necessary
- CDCL is method of choice for SAT (no contention)
  - Solves „well-behaved“ problems with millions of variables
  - Many „real“ problems are „well-behaved“: circuit and protocol verification, configuration problems, software verification.

History: the Davis-Putnam Algorithm (DP-1960)

- Martin Davis, Hilary Putnam: A Computing Procedure for Quantification Theory. *Journal of the ACM 7:201--215* (1960)
  - Context: proving theorems of predicate logic
    - Formula P is a contradiction iff there is a finite contradiction
    - Iteratively generate the Herbrand universe H_i and substitute into P
    - Each instance $P(H_i)$ is a propositional formula, then solve $SAT(P(H_i))$
- The DP-1960 algorithm consists of 3 rules
  1. One-Literal Rule (*Unit Propagation – UP*)
  2. Affirmative-Negative Rule (*Pure Literal – PL*)
  3. Elimination of conflicts (*Resolution*)
- Later, rule 3 was replaced by 3* (DPLL, 1962)

  3*. Splitting Rule (*Case distinction → SAT-Solving*)

History: the Davis-Putnam Algorithm (DP-1960)

- Rule 1 (One-Literal Clauses)
  a) F is inconsistent, if it contains two unit clauses {p} and {¬p}
  b) Else, if F contains a unit clause {p}, then delete all clauses containing p, and delete ¬p from all clauses.

     The result F´ is inconsistent iff F is inconsistent.
  a) The case of a unit clause {¬p} is analogous to (b).

  If F´ is empty, then F is consistent.
  - All clauses were deleted, hence all are satisfied.
- Rule 2 (Affirmative-Negative Rule)
  - If an atom p appears only positively (affirmative) or only negatively, then delete all clauses containing p.
  - The result F´ is inconsistent iff F is inconsistent.
  - If F´ is empty, then F is consistent.

History: the Davis-Putnam Algorithm (DP-1960)

- Rule 3 (Elimination of Atomic Formulas)
  - If an atom p appears ***both*** positively (in clause set A) ***and*** negatively (in clause set B), then

    form clause-sets A´ and B´ with $A = A' \vee p$ and $B = B' \vee \neg p$

    and rearrange F into $F = (A' \vee p) \wedge (B' \vee \neg p) \wedge R$,

    where p does not occur in A´, B´ and R.

    Now F is inconsistent iff $F' = (A' \vee B') \wedge R$ is inconsistent
  - Proof:

    F is inconsistent iff it is inconsistent for *both* p=0 *and* p=1.

    $F = A' \wedge R$ for p=0, and $F = B' \wedge R$ for p=1,

    hence F is inconsistent iff

    $F' = (A' \wedge R) \vee (B' \wedge R) = (A' \vee B') \wedge R$ is inconsistent.

History: the Davis-Putnam Algorithm (DP-1960)

- Implementation of Rule 3 (Eliminating Atomic Formulas)
  - Rearrange F into $F = (A' \vee p) \wedge (B' \vee \neg p) \wedge R$
  - F is inconsistent iff $F' = (A' \vee B') \wedge R$ is inconsistent
  - In short: factor out p and resolve on p. $(A' \vee B')$ consists of all resolvents between clauses in A and clauses in B.
    - Ex.: $(a \vee p) \wedge (b \vee p) \wedge (c \vee \neg p) \wedge (d \vee \neg p) = [(a \wedge b) \vee p] \wedge [(c \wedge d) \vee \neg p]$ Form $(a \wedge b) \vee (c \wedge d) = (a \vee c) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \vee d)$. These are exactly the resolvents. The parent clauses can be deleted: if $(a \wedge b)$ is satisfied in F´, then F is satisfied by additionally setting p=0, and analogously, if $(c \wedge d)$ is satisfied in F´, then we may set p=1 to satisfy F.
- Rule 3 yields a quantifier elimination (QE) procedure
  - $\exists p, x_1, \ldots, x_n : F \equiv \exists x_1, \ldots, x_n : F' \equiv \ldots . \equiv B$, where $B \in \{\top, \bot\}$
- Bad news: clauses get longer, and clause set explodes, and $F' = (A' \vee B') \wedge R$ is no longer in CNF, needs fresh CNF conversion

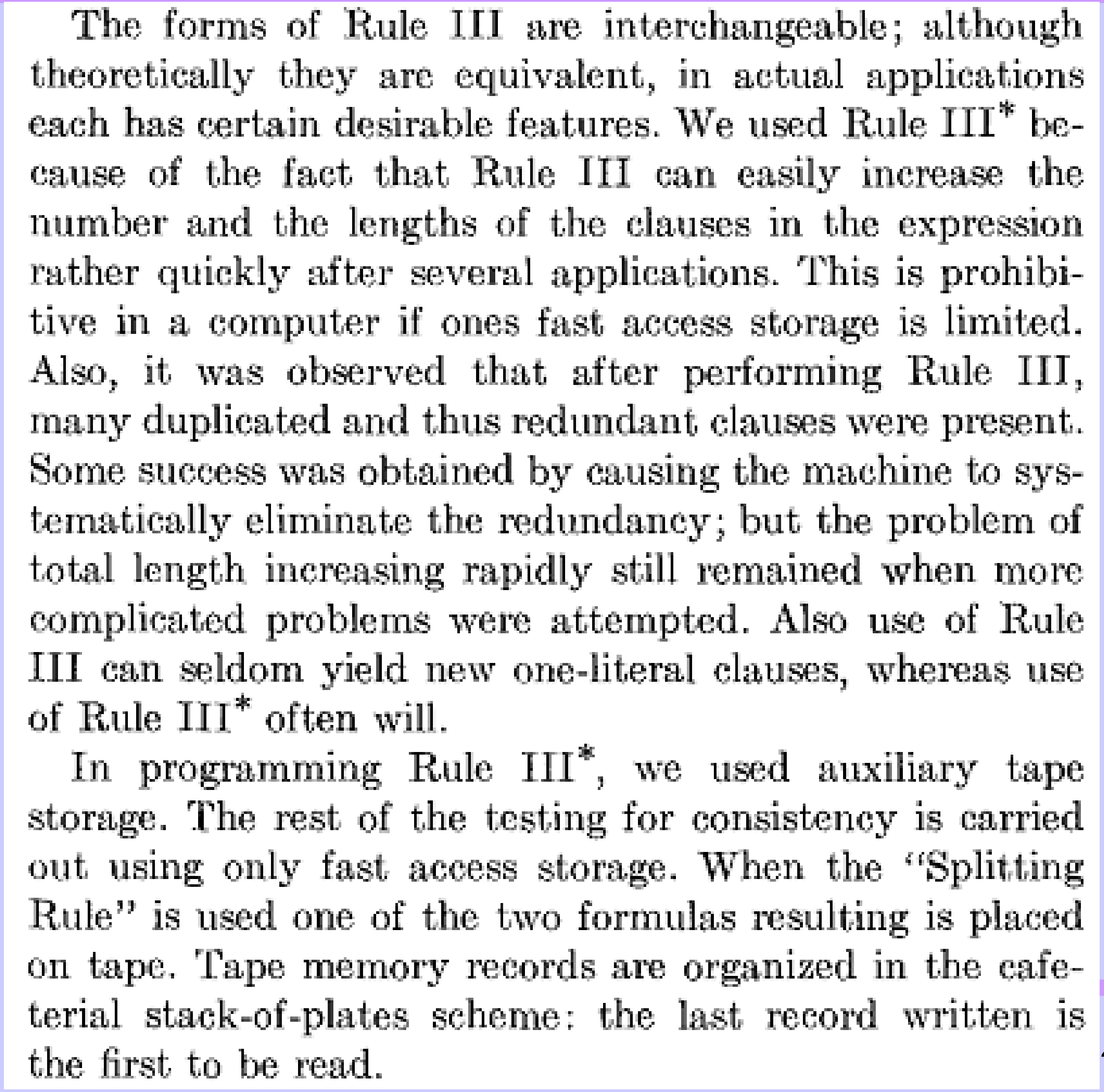
Example: the Davis-Putnam Algorithm (DP-1960)

- The DP-Algorithm consists of 3 rules
  1. One-Literal (Unit Propagation – UP)
  2. Affirmative-Negative (Pure Literal – PL)
  3. Elimination of conflicts (Resolution)
- Example
  - $S_0 = \{\{x, y, z\}, \{\neg x, y, z\}, \{\neg x\}, \{z, \neg y\}\}$
  - 1c (UP of ¬x): $S_1 = \{\{y, z\}, \{z, \neg y\}\}$
  - 3 (resolution on y): $S_2 = \{\{z\}, \{z\}\} = \{\{z\}\}$
  - 2 (PL of z}: $S_3 = \{ \}$, hence consistent.
- Rule 3 renders DP-1960 inefficient
  - In order to eliminate p from F, *ALL* resolvents over p have to be computed. This leads to an explosion of new clauses.

The Davis-Logemann-Loveland Algorithm (DPLL 1962)

- Martin Davis, George Logemann, and Donald Loveland. A Machine Program for Theorem Proving. *Communications of the ACM 5:394—397* (1962).
- Rule 3* (*Splitting Rule*, replaces Rule 3)
  - If p occurs both positively and negatively in F, rearrange F into F = (A ∨ p) ∧ (B ∨ ¬p) ∧ R, where p does not occur in R.
  - F is inconsistent iff *both* (A ∧ R) *and* (B ∧ R) are inconsistent
  - *Proof*: F is inconsistent iff it is inconsistent for *both* p=0 *and* for p=1. Now F= A ∧ R for p=0, and F= B ∧ R for p=1.
- Implementation of Rule 3*
  - Set p=1 and p=0 one after another in F (e.g. as unit clauses)
  - Do not form new clauses, instead perform unit propagation.
  - Clauses are shortened, sometimes eliminated, the problem is simplified, especially through unit propagation.

History: The Davis-Logemann-Loveland Algorithm



The forms of Rule III are interchangeable; although theoretically they are equivalent, in actual applications each has certain desirable features. We used Rule III* because of the fact that Rule III can easily increase the number and the lengths of the clauses in the expression rather quickly after several applications. This is prohibitive in a computer if ones fast access storage is limited. Also, it was observed that after performing Rule III, many duplicated and thus redundant clauses were present. Some success was obtained by causing the machine to systematically eliminate the redundancy; but the problem of total length increasing rapidly still remained when more complicated problems were attempted. Also use of Rule III can seldom yield new one-literal clauses, whereas use of Rule III* often will.

In programming Rule III*, we used auxiliary tape storage. The rest of the testing for consistency is carried out using only fast access storage. When the "Splitting Rule" is used one of the two formulas resulting is placed on tape. Tape memory records are organized in the cafeterial stack-of-plates scheme: the last record written is the first to be read.

The DPLL Algorithm (of 1962)

```
boolean DPLL(ClauseSet S){

    //1. Simplify S (unit constraint propagation)
    while (S contains a unit clause {$\ell$}) {
        delete from S clauses containing $\ell$;      // unit-subsumption
        delete $\neg\ell$ from all clauses in S          // unit-resolution mit subsumption
    }

    //2. Trivial case?
    if ($\square \in$S)  return false;                    // constraint unsatisfiable
    if (S=={}) return true;                        // nothing left to satisfy

    //3. Case split and recursion
    choose a literal $\ell$ occurring in S;            // Heuristic (intelligence) needed!
    if( DPLL(S $\cup$ {$\ell$}) ) return true;              // first recursive branch: try $\ell$ := true
    else if ( DPLL(S $\cup$ {$\neg\ell$}) ) return true;        // backtracking: try $\ell$ := false
    else return false;
}
```

Observations for DPLL in practice

- No deduction of new clauses
  - No dynamic storage allocation
- Algorithm lives (and dies) with unit propagation
  - UP dominates run-time in practice (> 90% UP).
  - This is necessarily so:
    - With 100 variables there are 2^{100} cases without UP
    - With complete UP there are only 99 propagations
    - Typical value in practice: 90 propagations, 2^{10} remaining cases
- Lesson from practice (and secret behind DPLL)
  - Only few decisions are essential, the remaining cases follow as immediate consequences, ruling out many theoretical alternatives.

Example: SAT-Solving with DPLL

- $S_0 = \{\{x, y, z\}, \{\neg x, y, z\}, \{\neg x\}, \{z, \neg y\}\}$
  - unit propagation of $\neg x$
    - $\{\neg x\}$ subsumes $\{\neg x, y, z\}$, hence $\neg x \wedge (\neg x \vee y \vee z) \equiv \neg x$
    - $\{\neg x\}$ unit-resolves with $\{x, y, z\}$ to $\{y, z\}$, and $\{y, z\}$ subsumes $\{x, y, z\}$
- $S_1 = \{\{y, z\}, \{z, \neg y\}\}$
  - Heuristically choose y as decision variable:
  - Case 1: let $y=1$
  - $S_2 = \{\{y\}, \{y, z\}, \{z, \neg y\}\}$
  - unit propagation of y yields $S_3 = \{\{z\}\}$,
  - unit propagation of z yields $S_4 = \{\ \}$, return true.

Variable Selection Heuristics for DPLL

- Some Heuristics for variable selection:
  - Coose the literal which occurs most often.
    - Then the formula is simplified in the most places
  - Choose a literal L from 2-clause (binary clause) {K, ¬L}.
    - Then K=1 if L=1, resp. L=0 if K=0, because clause encodes (L → K)
    - Clever choice of K resp. L immediately leads to UP, eliminating a decision
  - Choose a literal from a short(est) clause.
    - Will soon produce a binary clause, then a UP
  - For each literal L, compute how F would be shortened by UP after assigning L. Choose that L which has the greatest effect.
    - This simplifies F most before the next decision.

Correctness of DPLL

- $S|_L := \{C \setminus \{\neg L\} \mid C \in S, L \notin C\}$ reflects unit constraint propagation
  - unit clause $\{L\}$ subsumes all clauses C with $L \in C$.
  - unit clause $\{L\}$ resolves with all clauses $C = C_1 \cup \{\neg L\}$ giving C_1, and C_1 in turn subsumes C.
- If $\{L\} \in S$, then S is satisfiable iff $S|_L$ is satisfiable.
- S is satisfiable iff $S \cup \{\{L\}\}$ or $S \cup \{\{\neg L\}\}$ is satisfiable, for an arbitrary literal L.
- Termination: Number n of variables occurring in S (before Step 2) decreases with each rekcursive call, so finally either $\square \in S$ or $S = \{\ \}$.

First Step from DPLL to CDCL: Elementary Learning

- If F=0 after a sequence of variable assignments $x_i = b_i$, $b_i \in \{0, 1\}$, i.e. $\beta(F)=0$, then we have hit upon a root N of F.
  - N is given by a term $N=(\bigwedge\{x_i\} \bigwedge\{\neg y_i\})$, $\beta(x_i)=1$, $\beta(y_i)=0$
  - We have $N \models \neg F$ and hence $F \models \neg N$
- The negation ¬N yields a clause C
  - $F \models C$ implies $F \equiv F \wedge C$, i.e. C may be safely added to F.
  - We have „learned“ to avoid this root in the future.
- CDCL: conflict driven clause learning (first idea only)
  - Good: C need only contain the decision variables in N
  - Bad: in general, many decisions were taken before hitting the root, not all of them relevant for the root. C is still far too long, does not capture the root condition precisely, i.e. C is too special and of limited re-use.

Example: Elementary CDCL

- $S_0 = \{\{x, y\}, \{\neg y, z\}, \{\neg z, x\}\}$
  - Heuristically choose decision variable x :
  - Case 1: let x=0
  - $S_1 = \{\{\neg x\}, \{x, y\}, \{\neg y, z\}, \{\neg z, x\}\}$
  - unit propagation ¬x
  - $S_2 = \{\{y\}, \{\neg y, z\}, \{\neg z\}\}$
  - unit propagation y
  - $S_3 = \{\{z\}, \{\neg z\}\}$
  - unit propagation z
  - $S_4 = \{\{ \}\}$, return false
- We learn that $S_0 = 0$ if x=0, hence C = {x}.
  - If we restart SAT(S_0), we are immediatel led away from the root.

Example: Elementary CDCL

- Analysis of example $S_0 = \{\{x, y\}, \{\neg y, z\}, \{\neg z, x\}\}$

x	y	z	S_0
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

- From the previous variable assignments we learned: we hit the root N=(x=0, y=1, z=1).
- Since x was the only decision variable, we further learn that N is part of a cluster of 4 roots.

- In the future, the learned clause C = {x} directs us away from any of those 4 roots
  - E.g. after a fresh call and attempted propagations $\neg z$ und $\neg y$

Principle of Learning in CDCL

- Key insight: start learning process with *conflict clause* K
  - Conflict clause (failure clause) K is the clause which becomes empty in Step 2 of DPLL, i.e. β(K)=0, which is *always* due to a unit propagation in Step 1 (decisions never fail a clause).
  - As a consequence of a decision, some clause R („reason“-clause) became unit and forced L=1 for a literal L in R.
  - As an immediate consequence, K failed, i.e. the sole remaining literal L´ in unit clause K was forced to L´=0.
  - Hence L und L´ are complementary literals and R and K have a resolvent without this literal. Since K and R were unit, this resolvent is also a failure clause K´ (under the current assignment).
  - Now iterate the process …. (there are still some open details)

Example: Principle of Learning in CDCL

- $S_0 = \{\{x, y\}, \{\neg y, z\}, \{\neg z, x\}\}$
  - Heuristically choose decision variable x:
  - Case 1: Let x=0
  - $S_1 = \{\{\neg x\}, \{x, y\}, \{\neg y, z\}, \{\neg z, x\}\}$
  - unit propagation ¬x (reason for ¬x is decision x=0)
  - $S_2 = \{\{y\}, \{\neg y, z\}, \{\neg z\}\}$
  - unit propagation y (reason for y is $R_1=\{x, y\}$).
  - $S_3 = \{\{z\}, \{\neg z\}\}$.
  - unit propagation z (reason for z is $R_2=\{\neg y, z\}$))
  - $S_4 = \{\{\ \}\}$, return false (failure clause is $K=\{\neg z, x\}$)
- We have $\beta(\{\neg z, x\}) = 0$. Resolution with $R_2= \{\neg y, z\}$ yields $\{\neg y, x\}$, further resolution with $R_1= \{x, y\}$ yields $K´=\{x\}$.

Example: Principle of Learning in CDCL

- Analysis of Example $S_0 = \{\{x, y\}, \{\neg y, z\}, \{\neg z, x\}\}$

x	y	z	S_0
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

- From failure clause $K=\{\neg z, x\}$ we learn: we hit the *cluster* of roots $N=(x=0, z=1)$
- After resolution of K with $R_2=\{\neg y, z\}$ we learn $\{\neg y, x\}$, i.e. $N' = (x=0, y=1)$ is a ``neighboring´´ cluster of roots.
- After resolution with $R_1=\{x, y\}$ we learn $\{x\}$ with cluster $N'' = (x=0)$.

- Future hits into cluster $N'' = (x=0)$ (e.g. by a different UP sequence) are precluded by learned clause $K' = \{x\}$.

Summary: Decision Procedures for Propositional Logic

- Let $F \equiv (A \lor p) \land (B \lor \neg p) \land R$ (where p not in A, B, R)
  - This form can always be achieved
- Davis-Putnam (*elimination rule*): $F ≌ (A \lor B) \land R$
  - p is eliminated from $(A \lor B) \land R$
  - F is inconsistent iff $(A \lor B) \land R$ is inconsistent
    - Formula increases (very much) in size, but is strictly „simpler“ because p is eliminated.
- Davis-Logemann-Loveland *(splitting rule)*:
  - F is inconsistent iff both $(A \land R)$ and $(B \land R)$ are inconsistent
    - No fresh CNF conversion necesaary, formula is smaller, divide&conquer
- SAT-Solving (optimization of *splitting rule*):
  - F inconsistent iff both $F|_{p=0}$ and $F|_{p=1}$ inconsistent
- Resolution: $(A \lor p)\land(B \lor \neg p)\land R \equiv (A \lor p)\land(B \lor \neg p)\land(A \lor B)\land R$