By repeating this process, we will eventually obtain a clause that consists only of complements of decision literals and can be used in the “Backjump” rule.

Moreover, such a clause is a good candidate for learning.

Learning Clauses

The DPLL system can be extended by two rules to learn and to forget clauses:

Learn:

$M \parallel N \;\Rightarrow_{\text{DPLL}}\; M \parallel N \cup \{C\}$

if $N \models C$.

Forget:

$M \parallel N \cup \{C\} \;\Rightarrow_{\text{DPLL}}\; M \parallel N$

if $N \models C$.

If we ensure that no clause is learned infinitely often, then termination is guaranteed.

The other properties of the basic DPLL system hold also for the extended system.

Further Information

The ideas described so far heve been implemented in the SAT checker *zChaff*.

Further information:
Lintao Zhang and Sharad Malik: The Quest for Efficient Boolean Satisfiability Solvers, Proc. CADE-18, LNAI 2392, pp. 295–312, Springer, 2002.

Robert Nieuwenhuis, Albert Oliveras, Cesare Tinelli: Solvin SAT and SAT Modulo Theories: From an abstract Davis-Putnam-Logemann-Loveland precedure to DPLL(T), pp 937–977, Journal of the ACM, 53(6), 2006.

1.6 Splitting into Horn Clauses (Extra Topic)

- A *Horn clause* is a clause with at most one positive literal.
- They are typically denoted as implications: $P_1, \ldots, P_n \rightarrow Q$.
  (In general we can write $P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m$ for $\neg P_1 \vee \ldots \vee \neg P_n \vee Q_1 \vee \ldots \vee Q_m$.)
- Compared to arbitrary clause sets, Horn clause sets enjoy further properties:
  - Horn clause sets have unique minimal models.
  - Checking satisfiability is often of lower complexity.

Propositional Horn Clause SAT is in $\mathbf{P}$

boolean HornSAT(literal set M, Horn clause set N) {
    if (all clauses in N are supported by M) return true;
    elsif (a negative clause in N is not supported by M) return false;
    elsif (N contains clause $P_1, \ldots, P_n \rightarrow Q$ where
            $\{P_1, \ldots, P_n\} \subseteq M$ and $Q \notin M$)
        return HornSAT($M \cup \{Q\}$, N);
}

A clause $P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m$ is *supported* by M if $\{P_1, \ldots, P_n\} \not\subseteq M$ or some $Q_i \in M$. A *negative* clause consists of negative literals only.

Initially, HornSAT is called with an empty literal set M.

Lemma 1.13 *Let N be a set of propositional Horn clauses.Then:*

(1) HornSAT($\emptyset$, N)=true iff N is satisfiable

(2) HornSAT is in $\mathbf{P}$

Proof. (1) (Idea) For example, by induction on the number of positive literals in N.

(2) (Scetch) For each recursive call M contains one more positive literal. Thus HornSAT terminates after at most n recursive calls, where n is the number of propositional variables in N. □

SplitHornSAT

boolean SplitHornSAT(clause set N) {
  if (N is Horn)
g   return HornSAT($\emptyset$,N);
  else {
    select non Horn clause $P_1, \dots, P_n \rightarrow Q_1, \dots, Q_m$ from N;
    $N' = N \setminus \{P_1, \dots, P_n \rightarrow Q_1, \dots, Q_m\}$;
    if (SplitHornSAT($N' \cup \{P_1, \dots, P_n \rightarrow Q_1\}$)) return true;
    else return
      SplitHornSAT($N' \cup \{\rightarrow Q_2, \dots, Q_m\} \cup \bigcup_i \{\rightarrow P_i\} \cup \{Q_1 \rightarrow\}$);
  }
}

Lemma 1.14 *Let N be a set of propositional clauses. Then:*

(1) SplitHornSAT(N)=true iff N is satisfiable

(2) SplitHornSAT(N) terminates

Proof. (1) (Idea) Show that N is satisfiable iff $N' \cup \{P_1, \dots, P_n \rightarrow Q_1\}$ is satisfiable or $N' \cup \{\rightarrow Q_2, \dots, Q_m\} \cup \bigcup_i \{\rightarrow P_i\} \cup \{Q_1 \rightarrow\}$ is satisfiable for some clause $P_1, \dots, P_n \rightarrow Q_1, \dots, Q_m$ from N.

(2) (Idea) Each recursive call reduces the number of positive literals in non Horn clauses.
$\square$

1.7 Other Calculi

OBDDs (Ordered Binary Decision Diagrams):

Minimized graph representation of decision trees, based on a fixed ordering on propositional variables,

see script of the Computational Logic course,

see Chapter 6.1/6.2 of Michael Huth and Mark Ryan: *Logic in Computer Science: Modelling and Reasoning about Systems*, Cambridge Univ. Press, 2000.

FRAIGs (Fully Reduced And-Inverter Graphs)

Minimized graph representation of boolean circuits.

1.8 Example: SUDOKU

	1	2	3	4	5	6	7	8	9
1								1	
2	4								
3		2							
4					5		4		7
5			8				3		
6			1		9				
7	3			4			2		
8		5		1					
9				8		6			

Idea: $p^d_{i,j}$=true iff the value of square i, j is d

For example: $p^8_{3,5} = true$

Coding SUDOKU by propositional clauses

- Concrete values result in units: $p^d_{i,j}$
- For every value, column we generate: $\neg p^d_{i,j} \vee \neg p^d_{i,j+k}$
  Accordingly for all rows and 3×3 boxes
- For every square we generate: $p^1_{i,j} \vee \ldots \vee p^9_{i,j}$
- For every two different values, square we generate: $\neg p^d_{i,j} \vee \neg p^{d'}_{i,j}$
- For every value, column we generate: $p^d_{i,0} \vee \ldots \vee p^d_{i,9}$
  Accordingly for all rows and 3×3 boxes

Constraint Propagation is Unit Propagation

	1	2	3	4	5	6	7	8	9
1								1	
2	4								
3		2							
4					5		4		7
5			8				3		
6			1		9				
7	3			4	**7**		2		
8		5		1					
9				8		6			

From $\neg p^3_{1,7} \vee \neg p^3_{5,7}$ and $p^3_{1,7}$ we obtain by unit propagating $\neg p^3_{5,7}$ and further from $p^1_{5,7} \vee p^2_{5,7} \vee p^3_{5,7} \vee p^4_{5,7} \vee \ldots \vee p^9_{5,7}$ we get $p^1_{5,7} \vee p^2_{5,7} \vee p^4_{5,7} \vee \ldots \vee p^9_{5,7}$.

2 Linear Arithmetic (LA)

We consider boolean combinations of linear arithmetic atoms such as $3.5x - 4y \geq 7$ and search rational values for the variables x, y such that the disequation holds.

2.1 Syntax

Syntax:

- non-logical symbols (domain-specific) (e.g. $x, +$, values from $\mathbb{Q}, \geq$)
  $\Rightarrow$ terms, atomic formulas
- logical symbols (domain-independent) (e.g. $\wedge, \rightarrow$)
  $\Rightarrow$ Boolean combinations (no quantification)

Signature

A signature

$$\Sigma = (\Omega, \Pi),$$

fixes an alphabet of non-logical symbols, where

- Ω is a set of *function symbols* f with *arity* $n \geq 0$, written $\mathsf{arity}(f) = n$,
- Π is a set of *predicate symbols* p with arity $m \geq 0$, written $\mathsf{arity}(p) = m$.

The linear arithmetic signature is

$$\Sigma_{\text{LA}} = (\mathbb{Q} \cup \{+, -, *\}, \{\geq, \leq, >, <\})$$

Variables

Linear arithmetic admits the formulation of abstract, schematic assertions. (Object) variables are the technical tool for schematization.

We assume that

$$X$$

is a given countably infinite set of symbols which we use for (the denotation of) *variables*.

Context-Free Grammars

We define many of our notions on the bases of context-free grammars. Recall, that a context-free grammar $G = (N, T, P, S)$ consists of:

- a set of non-terminal symbols N
- a set of terminal symbols T
- a set P of rules $A ::= w$ where $A \in N$ and $w \in (N \cup T)^*$
- a start symbol S where $S \in N$

For rules $A ::= w_1$, $A ::= w_2$ we write $A ::= w_1 \mid w_2$

Terms

Terms over Σ_{LA} (resp., Σ_{LA}-terms) are formed according to these syntactic rules:

$$\begin{array}{lcll} s, t, u, v & ::= & x \mid q * x \mid q \quad , x \in X, q \in \mathbb{Q} & \text{(variable, rational)} \\ & & \mid s + t \mid s - t & \text{(sum, difference)} \end{array}$$

By $\mathrm{T}_{\Sigma_{\mathrm{LA}}}(X)$ we denote the set of Σ_{LA}-terms (over X). A term not containing any variable is called a *ground term*. By $\mathrm{T}_{\Sigma_{\mathrm{LA}}}$ we denote the set of Σ_{LA}-ground terms.

Atoms

Atoms (also called atomic formulas) over Σ_{LA} are formed according to this syntax:

$$\begin{array}{lcll} A, B & ::= & s \geq t \mid s \leq t \quad , s, t \in \mathrm{T}_{\Sigma_{\mathrm{LA}}}(X) & \text{(non-strict)} \\ & & \mid s > t \mid s < t \quad , s, t \in \mathrm{T}_{\Sigma_{\mathrm{LA}}}(X) & \text{(strict)} \end{array}$$

Quantifier Free Formulas

$\mathrm{QF}_{\Sigma_{\mathrm{LA}}}(X)$ is the set of positive boolean formulas over Σ_{LA} defined as follows:

$$\begin{array}{lcll} F, G, H & ::= & \bot & \text{(falsum)} \\ & \mid & \top & \text{(verum)} \\ & \mid & A & \text{(atomic formula)} \\ & \mid & \neg F & \text{(negation)} \\ & \mid & (F \wedge G) & \text{(conjunction)} \\ & \mid & (F \vee G) & \text{(disjunction)} \\ & \mid & (F \rightarrow G) & \text{(implication)} \\ & \mid & (F \leftrightarrow G) & \text{(equivalence)} \end{array}$$

Linear Arithmetic Semantics

The Σ_{LA}-*algebra* (also called Σ_{LA}-interpretation or Σ_{LA}-structure) is the triple

$$\mathcal{A}_{\text{LA}} = (\mathbb{Q},\ (+_{\mathcal{A}_{\text{LA}}}, -_{\mathcal{A}_{\text{LA}}}, *_{\mathcal{A}_{\text{LA}}}),\ (\leq_{\mathcal{A}_{\text{LA}}}, \geq_{\mathcal{A}_{\text{LA}}}, <_{\mathcal{A}_{\text{LA}}}, >_{\mathcal{A}_{\text{LA}}}))$$

where $+_{\mathcal{A}_{\text{LA}}}, -_{\mathcal{A}_{\text{LA}}}, *_{\mathcal{A}_{\text{LA}}}, \leq_{\mathcal{A}_{\text{LA}}}, \geq_{\mathcal{A}_{\text{LA}}}, <_{\mathcal{A}_{\text{LA}}}, >_{\mathcal{A}_{\text{LA}}}$ are the “standard” intepretations of $+, -, *, \leq, \geq, <, >$, respectively.

Linear Arithmetic Assignments

A variable has no intrinsic meaning. The meaning of a variable has to be defined externally (explicitly or implicitly in a given context) by an assignment.

A *(variable) assignment*, also called a *valuation* for linear arithmetic is a map $\beta : X \to \mathbb{Q}$.

Truth Value of a Formula with Respect to β

$\mathcal{A}_{\text{LA}}(\beta) : \text{QF}_{\Sigma_{\text{LA}}}(X) \to \{0, 1\}$ is defined inductively as follows:

$$\begin{aligned}
\mathcal{A}_{\text{LA}}(\beta)(\bot) &= 0 \\
\mathcal{A}_{\text{LA}}(\beta)(\top) &= 1 \\
\mathcal{A}_{\text{LA}}(\beta)(s \sharp t) = 1 \quad &\Leftrightarrow \quad (\mathcal{A}_{\text{LA}}(\beta)(s) \sharp_{\mathcal{A}_{\text{LA}}} \mathcal{A}_{\text{LA}}(\beta)(t)) \\
&\sharp \in \{\leq, \geq, <, >\} \\
\mathcal{A}_{\text{LA}}(\beta)(\neg F) = 1 \quad &\Leftrightarrow \quad \mathcal{A}_{\text{LA}}(\beta)(F) = 0 \\
\mathcal{A}_{\text{LA}}(\beta)(F \rho G) &= \mathsf{B}_\rho(\mathcal{A}_{\text{LA}}(\beta)(F), \mathcal{A}_{\text{LA}}(\beta)(G)) \\
&\text{with } \mathsf{B}_\rho \text{ the Boolean function associated with } \rho
\end{aligned}$$

$\mathcal{A}_{\text{LA}}(\beta)(x) = \beta(x)$, $\mathcal{A}_{\text{LA}}(\beta)(s \circ t) = \mathcal{A}_{\text{LA}}(\beta)(s) \circ_{\mathcal{A}_{\text{LA}}} \mathcal{A}_{\text{LA}}(\beta)(t)$, $\circ \in \{+, -, *\}$, $\mathcal{A}_{\text{LA}}(\beta)(q) = q$ for all $q \in \mathbb{Q}$.

2.2 Models, Validity, and Satisfiability

F is *valid* in $\mathcal{A}_{\text{LA}}$ under assignment β:

$$\mathcal{A}_{\text{LA}}, \beta \models F \quad :\Leftrightarrow \quad \mathcal{A}_{\text{LA}}(\beta)(F) = 1$$

F is *valid* in $\mathcal{A}_{\text{LA}}$ ($\mathcal{A}_{\text{LA}}$ is a *model* of F):

$$\mathcal{A}_{\text{LA}} \models F \quad :\Leftrightarrow \quad \mathcal{A}_{\text{LA}}, \beta \models F, \text{ for all } \beta \in X \to \mathbb{Q}$$

F is called *satisfiable* iff there exist a β such that $\mathcal{A}_{\text{LA}}, \beta \models F$. Otherwise F is called *unsatisfiable*.

On Quantification

Linear arithmetic can also be considered with respect to quantification. The quantifiers are $\exists$ meaning "there exists" and $\forall$ meaning "for all". For example, $\exists x\,(x \geq 0)$ is valid (or true) in $\mathcal{A}_{\mathrm{LA}}$, $\forall x\,(x \geq 0)$ is unsatisfiable (or false) and $\forall x\,(x \geq 0 \lor x < 0)$ is again valid.

Note that a quantifier free formula is satisfiable iff the existential closure of the formula is valid. If we introduce new free constants c_i for the variables x_i of a quantifier free formula, where $\mathcal{A}_{\mathrm{LA}}(c_i) = q_i$ for some $q_i \in \mathbb{Q}$, then a quantifier free formula is satisfiable iff the same formula where variables are replaced by new free constants is satisfiable.

Some Important LA Equivalences

Proposition 2.1 *The following equivalences are valid for all LA terms s, t:*

$$\begin{array}{ll} \neg s \geq t \leftrightarrow s < t & \\ \neg s \leq t \leftrightarrow s > t & \text{(Negation)} \end{array}$$

$$(s = t) \leftrightarrow (s \leq t \land s \geq t) \quad \text{(Equality)}$$

$$\begin{array}{ll} s \geq t \leftrightarrow t \leq s & \\ s > t \leftrightarrow t < s & \text{(Swap)} \end{array}$$

With $\lesssim$ we abbreviate $<$ or $\leq$.

The Fourier-Motzkin Procedure

boolean FM(Set N of LA atoms) {
  if ($N = \emptyset$) return true;
  elsif (N is ground) return $\mathcal{A}_{\mathrm{LA}}(N)$;
  else {
    select a variable x from N;
    transform all atoms in N containing x into $s_i \lesssim x$, $x \lesssim t_j$
    and the subset N' of atoms not containing x;
    compute $N^* := \{s_i \lesssim_{i,j} t_j \mid s_i \lesssim_i x \in N,\ x \lesssim_j t_j \in N \text{ for all } i, j\}$
    where $\lesssim_{i,j}$ is strict iff at least one of $\lesssim_i$, $\lesssim_j$ is strict
    return FM($N' \cup N^*$);
  }
}

Properties of the Fourier-Motzkin Procedure

- Any ground set N of linear arithmetic atoms can be easily decided.
- FM(N) terminates on any N as in recursive calls N has strictly less variables.
- The set $N' \cup N^*$ is worst case of size $O(|N|^2)$.
- FM(N)=true iff N is satisfiable in $\mathcal{A}_{\mathrm{LA}}$.
- The procedure was invented by Fourier (1826), forgotten, and then rediscovered by Dines (1919) and Motzkin (1936).
- There are more efficient methods known, e.g., the simplex algorithm.

2.3 The DPLL(T) Procedure

Goal:
Given a propositional formula in CNF (or alternatively, a finite set N of clauses), where the atoms represent ground formulas over some theory T, check whether it is satisfiable in T. (and optionally: output *one* solution, if it is satisfiable).

Assumption:
Again, clauses contain neither duplicated literals nor complementary literals.

Remark:
We will use LA as an ongoing example for T and consider DPLL(LA).

Notions with Respect to the Theory T

If a partial valuation M is T-consistent and F a formula such that $M \models F$, then we say that M *is a T-model of F*.

If F and G are formulas then F *entails* G *in* T, written $F \models_T G$ if $F \wedge \neg G$ is T-inconsistent.

Example: $x > 1 \not\models x > 0$ but $x > 1 \models_{\mathrm{LA}} x > 0$

Remark

M stands again for a list of propositional literals. As every propositional literal stands for a ground formula T, there are actually two interpretations of M. We write $M \models F$ if F is entailed by M propositionally. We write $M \models_T F$ if the T ground formulas represented by M entail F.