1.9 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 $\neg P_1 \lor \ldots \lor \neg P_n \lor Q_1 \lor \ldots \lor Q_m$.)
- 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 P**

```java
boolean HornSAT(literal set $M$, Horn clause set $N$) {
    if (all clauses in $N$ are supported by $M$) return true;
    elseif (a negative clause in $N$ is not supported by $M$) return false;
    elseif ($N$ 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 1.18** Let $N$ be a set of propositional Horn clauses. Then:

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

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

**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 $N$. \qed

---

24
SplitHornSAT

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

Lemma 1.19 Let \( N \) be a set of propositional clauses. Then:

1. \( \text{SplitHornSAT}(N) = true \) iff \( N \) is satisfiable
2. \( \text{SplitHornSAT}(N) \) terminates

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

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

1.10 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,


FRAIGs (Fully Reduced And-Inverter Graphs)

Minimized graph representation of boolean circuits.
1.11 Example: SUDOKU

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Idea: \( p_{i,j}^d = \text{true} \) iff the value of square \( i, j \) is \( d \)

For example: \( p_{3,5}^8 = \text{true} \)

Coding SUDOKU by propositional clauses

- Concrete values result in units: \( p_{i,j}^d \)
- For every value, column we generate: \( \neg p_{i,j}^d \lor \neg p_{i,j+k}^d \)
  Accordingly for all rows and \( 3 \times 3 \) boxes
- For every square we generate: \( p_{i,j}^1 \lor \ldots \lor p_{i,j}^9 \)
- For every two different values, square we generate: \( \neg p_{i,j}^d \lor \neg p_{i,j}^{d'} \)
- For every value, column we generate: \( p_{i,0}^d \lor \ldots \lor p_{i,9}^d \)
  Accordingly for all rows and \( 3 \times 3 \) boxes

Constraint Propagation is Unit Propagation

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

From \( \neg p_{5,7}^3 \lor \neg p_{5,7}^2 \) and \( p_{1,7}^1 \) we obtain by unit propagating \( \neg p_{5,7}^3 \) and further from \( p_{5,7}^1 \lor p_{5,7}^2 \lor p_{5,7}^3 \lor \ldots \lor p_{5,7}^9 \) we get \( p_{5,7}^1 \lor p_{5,7}^2 \lor p_{5,7}^3 \lor \ldots \lor p_{5,7}^9 \).
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. $\land, \rightarrow$)  
  $\Rightarrow$ Boolean combinations (no quantification)

Signature

A signature

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

fixes an alphabet of non-logical symbols, where
- $\Omega$ is a set of function symbols $f$ with arity $n \geq 0$, written $\text{arity}(f) = n$,
- $\Pi$ is a set of predicate symbols $p$ with arity $m \geq 0$, written $\text{arity}(p) = m$.

The linear arithmetic signature is

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

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 \( \Sigma_{\text{LA}} \) (resp., \( \Sigma_{\text{LA}} \)-terms) are formed according to these syntactic rules:

\[
\begin{align*}
s, t, u, v & ::= x \mid q \cdot x \mid q, x \in X, q \in \mathbb{Q} \text{ (variable, rational)} \\
& \mid s + t \mid s - t \text{ (sum, difference)}
\end{align*}
\]

By \( T_{\Sigma_{\text{LA}}}(X) \) we denote the set of \( \Sigma_{\text{LA}} \)-terms (over \( X \)). A term not containing any variable is called a ground term. By \( T_{\Sigma_{\text{LA}}} \) we denote the set of \( \Sigma_{\text{LA}} \)-ground terms.

Atoms

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

\[
\begin{align*}
A, B & ::= s \geq t \mid s \leq t , s, t \in T_{\Sigma_{\text{LA}}}(X) \text{ (non-strict)} \\
& \mid s > t \mid s < t , s, t \in T_{\Sigma_{\text{LA}}}(X) \text{ (strict)}
\end{align*}
\]

Quantifier Free Formulas

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

\[
\begin{align*}
F, G, H & ::= \bot \text{ (falsum)} \\
& \mid \top \text{ (verum)} \\
& \mid A \text{ (atomic formula)} \\
& \mid \neg F \text{ (negation)} \\
& \mid (F \land G) \text{ (conjunction)} \\
& \mid (F \lor G) \text{ (disjunction)} \\
& \mid (F \rightarrow G) \text{ (implication)} \\
& \mid (F \leftrightarrow G) \text{ (equivalence)}
\end{align*}
\]

28
Linear Arithmetic Semantics

The $\Sigma_{LA}$-algebra (also called $\Sigma_{LA}$-interpretation or $\Sigma_{LA}$-structure) is the triple

$$A_{LA} = (\mathbb{Q}, (+_{LA}, -_{LA}, \cdot_{LA}), (\leq_{LA}, \geq_{LA}, <_{LA}, >_{LA}))$$

where $+, -, \cdot, \leq, \geq, <, >$ are the “standard” interpretations of $+, -, \cdot, \leq, \geq, <, >$, 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 $\beta : X \rightarrow \mathbb{Q}$.

Truth Value of a Formula with Respect to $\beta$

$A_{LA}(\beta) : QF_{\Sigma_{LA}}(X) \rightarrow \{0, 1\}$ is defined inductively as follows:

- $A_{LA}(\beta)(\bot) = 0$
- $A_{LA}(\beta)(\top) = 1$
- $A_{LA}(\beta)(s \# t) = 1 \iff (A_{LA}(\beta)(s) \#_{LA} A_{LA}(\beta)(t))$, $\# \in \{\leq, \geq, <, >\}$
- $A_{LA}(\beta)(\neg F) = 1 \iff A_{LA}(\beta)(F) = 0$
- $A_{LA}(\beta)(F \rho G) = B_{\rho}(A_{LA}(\beta)(F), A_{LA}(\beta)(G))$
  with $B_{\rho}$ the Boolean function associated with $\rho$

$A_{LA}(\beta)(x) = \beta(x)$, $A_{LA}(\beta)(s \circ t) = A_{LA}(\beta)(s) \circ_{LA} A_{LA}(\beta)(t)$, $\circ \in \{+, -, \cdot\}$, $A_{LA}(\beta)(q) = q$ for all $q \in \mathbb{Q}$.

2.2 Models, Validity, and Satisfiability

$F$ is valid in $A_{LA}$ under assignment $\beta$:

$$A_{LA}, \beta \models F \iff A_{LA}(\beta)(F) = 1$$

$F$ is valid in $A_{LA}$ ($A_{LA}$ is a model of $F$):

$$A_{LA} \models F \iff A_{LA}, \beta \models F, \text{ for all } \beta \in X \rightarrow \mathbb{Q}$$

$F$ is called satisfiable iff there exist a $\beta$ such that $A_{LA}, \beta \models F$. 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 \( A_{LA} \), \( \forall x (x \geq 0) \) 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 \( A_{LA}(c_i) = q_i \) 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 \( s, t \):

\[
\neg s \geq t \leftrightarrow s < t \\
\neg s \leq t \leftrightarrow s > t \quad \text{(Negation)}
\]

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

\[
s \geq t \leftrightarrow t \leq s \\
s > t \leftrightarrow t < s \quad \text{(Swap)}
\]

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 \( A_{LA}(N) \);
    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 \( N^* := \{ s_i \lesssim_{i,j} t_j \mid s_i \lesssim x \in N, x \lesssim t_j \in N \text{ for all } i, j \} \)
        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.
• \( \text{FM}(N) \) terminates on any \( N \) as in recursive calls \( N \) has strictly less variables.
• The set \( N' \cup N^* \) is worst case of size \( O(|N|^2) \).
• \( \text{FM}(N)=\text{true} \) iff \( N \) is satisfiable in \( A_{\text{LA}} \).
• 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 \( \text{LA} \) as an ongoing example for \( T \) and consider DPLL(LA).

On \( \text{LA} \) as a Theory

We consider a specific formula language together with a satisfiability check for conjunctions of atoms (literals) as a theory \( T \). Note that a valuation \( M \) is interpreted as the conjunction of its literals.

Later on we will introduce theory notions based on sets of formulas or models.

For \( \text{LA} \) we consider the language defined before and Fourier-Motzkin as the satisfiability check for conjunctions of atoms. Variables in formulas without quantification can actually be considered as constants.

Notions with Respect to the Theory \( T \)

If a partial valuation \( M \) is \( T \)-consistent (satisfiable) 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 \( F \land \neg G \) is \( T \)-inconsistent.

Example: \( x > 1 \not\models x > 0 \) but \( x > 1 \models_{\text{LA}} x > 0 \)