**DPLL(T) Rules from DPLL**

**Unit Propagate:**

\[ M \parallel N \cup \{C \lor L\} \Rightarrow_{\text{DPLL(T)}} M L \parallel N \cup \{C \lor L\} \]

if \( C \) is false under \( M \) and \( L \) is undefined under \( M \).

**Decide:**

\[ M \parallel N \Rightarrow_{\text{DPLL(T)}} M L^d \parallel N \]

if \( L \) is undefined under \( M \).

**Fail:**

\[ M \parallel N \cup \{C\} \Rightarrow_{\text{DPLL(T)}} \text{fail} \]

if \( C \) is false under \( M \) and \( M \) contains no decision literals.

**Specific DPLL(T) Rules**

**T-Backjump:**

\[ M L^d M' \parallel N \cup \{C\} \Rightarrow_{\text{DPLL(T)}} M L' \parallel N \cup \{C\} \]

if \( M L^d M' \models \neg C \)

there is some “backjump clause” \( C' \lor L' \) such that

\( N \cup \{C\} \models_{T} C' \lor L' \) and \( M \models \neg C' \)

\( L' \) is undefined under \( M' \), and

\( L' \) or \( \overline{L'} \) occurs in \( N \) or in \( M L^d M' \).

**T-Learn:**

\[ M \parallel N \Rightarrow_{\text{DPLL(T)}} M \parallel N \cup \{C\} \]

if \( N \models_{T} C \) and each atom of \( C \) occurs in \( N \) or \( M \).

**T-Forget:**

\[ M \parallel N \cup \{C\} \Rightarrow_{\text{DPLL(T)}} M \parallel N \]

if \( N \models_{T} C \).

**T-Propagate:**

\[ M \parallel N \Rightarrow_{\text{DPLL(T)}} M L \parallel N \]

if \( M \models_{T} L \) where \( L \) is undefined in \( M \) and \( L \) or \( \overline{L} \) occurs in \( N \).
**DPLL(T) Properties**

The DPPL modulo theories system DPLL(T) consists of the rules Decide, Fail, Unit-Propagate, $T$-Propagate, $T$-Backjump, $T$-Learn and $T$-Forget.

The Lemma 1.9 and the Lemma 1.10 from DPLL hold accordingly for DPLL(T). Again we will reconsider termination when the needed notions on orderings are established.

**Lemma 2.2** If $\emptyset \parallel N \Rightarrow^{*}_{DPLL(T)} M \parallel N'$ and there is some conflicting clause in $M \parallel N'$, that is, $M \models \neg C$ for some clause $C$ in $N$, then either Fail or $T$-Backjump applies to $M \parallel N'$.

**Proof.** As in Lemma 1.11. \qed

**Lemma 2.3** If $\emptyset \parallel N \Rightarrow^{*}_{DPLL(T)} M \parallel N'$ and $M$ is $T$-inconsistent, then either there is a conflicting clause in $M \parallel N'$, or else $T$-Learn applies to $M \parallel N'$, generating a conflicting clause.

**Proof.** If $M$ is $T$-inconsistent, then there exists a subsequence $(L_1, \ldots, L_n)$ of $M$ such that $\emptyset \models_T L_1 \lor \ldots \lor L_n$. Hence the conflicting clause $L_1 \lor \ldots \lor L_n$ is either in $M \parallel N'$, or else it can be learned by one $T$-Learn step. \qed

**3 First-Order Logic**

First-order logic

- formalizes fundamental mathematical concepts
- is expressive (Turing-complete)
- is not too expressive (e.g. not axiomatizable: natural numbers, uncountable sets)
- has a rich structure of decidable fragments
- has a rich model and proof theory

First-order logic is also called (first-order) predicate logic.
3.1 Syntax

Syntax:

- non-logical symbols (domain-specific) ⇒ terms, atomic formulas
- logical symbols (domain-independent) ⇒ Boolean combinations, quantifiers

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 $$\text{arity}(p) = m.$$ If $$n = 0$$ then $$f$$ is also called a constant (symbol).
If $$m = 0$$ then $$p$$ is also called a propositional variable.
We use letters $$P, Q, R, S,$$ to denote propositional variables.
Refined concept for practical applications:
many-sorted signatures (corresponds to simple type systems in programming languages); not so interesting from a logical point of view.

Variables

Predicate logic 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 $\Sigma$ (resp., $\Sigma$-terms) are formed according to these syntactic rules:

$$s, t, u, v ::= x \quad (x \in X \quad \text{variable})$$
$$\mid f(s_1, \ldots, s_n) \quad (f \in \Omega, \text{arity}(f) = n \quad \text{functional term})$$

By $T_\Sigma(X)$ we denote the set of $\Sigma$-terms (over $X$). A term not containing any variable is called a ground term. By $T_\Sigma$ we denote the set of $\Sigma$-ground terms.

In other words, terms are formal expressions with well-balanced brackets which we may also view as marked, ordered trees. The markings are function symbols or variables. The nodes correspond to the subterms of the term. A node $v$ that is marked with a function symbol $f$ of arity $n$ has exactly $n$ subtrees representing the $n$ immediate subterms of $v$.

Atoms

Atoms (also called atomic formulas) over $\Sigma$ are formed according to this syntax:

$$A, B ::= p(s_1, \ldots, s_m) \quad (p \in \Pi, \text{arity}(p) = m)$$
$$\mid (s \approx t) \quad \text{(equation)}$$

Whenever we admit equations as atomic formulas we are in the realm of first-order logic with equality. Admitting equality does not really increase the expressiveness of first-order logic, (cf. exercises). But deductive systems where equality is treated specifically can be much more efficient.
Literals

$L ::= A$ (positive literal) \\
| $\neg A$ (negative literal)

Clauses

$C, D ::= \bot$ (empty clause) \\
| $L_1 \lor \ldots \lor L_k$, $k \geq 1$ (non-empty clause)

General First-Order Formulas

$F_{\Sigma}(X)$ is the set of first-order formulas over $\Sigma$ defined as follows:

$F, G, H ::= \bot$ (falsum) \\
| $\top$ (verum) \\
| $A$ (atomic formula) \\
| $\neg F$ (negation) \\
| $(F \land G)$ (conjunction) \\
| $(F \lor G)$ (disjunction) \\
| $(F \rightarrow G)$ (implication) \\
| $(F \leftrightarrow G)$ (equivalence) \\
| $\forall x F$ (universal quantification) \\
| $\exists x F$ (existential quantification)

Positions in terms, formulas

Positions of a term $s$ (formula $F$):

$\text{pos}(x) = \{ \varepsilon \}$, \\
$\text{pos}(f(s_1, \ldots, s_n)) = \{ \varepsilon \} \cup \bigcup_{i=1}^{n}\{ ip \mid p \in \text{pos}(s_i) \}$. \\
$\text{pos}(\forall x F) = \{ \varepsilon \} \cup \{ 1p \mid p \in \text{pos}(F) \}$ \\
Analogously for all other formulas.

Prefix order for $p, q \in \text{pos}(s)$:

$p$ above $q$: $p \leq q$ if $pp' = q$ for some $p'$, \\
$p$ strictly above $q$: $p < q$ if $p \leq q$ and not $q \leq p$, \\
$p$ and $q$ parallel: $p \parallel q$ if neither $p \leq q$ nor $q \leq p$.

Subterm of $s$ ($F$) at a position $p \in \text{pos}(s)$:

$s/\varepsilon = s$, \\
$f(s_1, \ldots, s_n)/ip = s_i/p$. 

31
Analougously for formulas \( (F/p) \).

Replacement of the subterm at position \( p \in \text{pos}(s) \) by \( t \):

\[
\begin{align*}
\delta[s[t]_e] &= t, \\
f(s_1, \ldots, s_n)[t]_{ip} &= f(s_1, \ldots, s_i[t]_p, \ldots, s_n).
\end{align*}
\]

Analougously for formulas \( (F[G]_p) \).

Size of a term \( s \):

\[
|s| = \text{cardinality of pos}(s).
\]

**Notational Conventions**

We omit brackets according to the following rules:

- \( \neg \), \( \lor \), \( \land \), \( \rightarrow \), \( \leftrightarrow \) (binding precedences)
- \( \lor \) and \( \land \) are associative and commutative
- \( \rightarrow \) is right-associative

\( Qx_1, \ldots, x_n F \) abbreviates \( Qx_1 \ldots Qx_n F \).

We use infix-, prefix-, postfix-, or mixfix-notation with the usual operator precedences.

Examples:

\[
\begin{align*}
s + t \ast u & \quad \text{for} \quad +(s, \ast(t, u)) \\
s \ast u \leq t + v & \quad \text{for} \quad \leq (*s, (+t, v)) \\
-s & \quad \text{for} \quad -(s) \\
0 & \quad \text{for} \quad 0()
\end{align*}
\]

**Example: Peano Arithmetic**

\[
\Sigma_{PA} = (\Omega_{PA}, \Pi_{PA})
\]

\[
\begin{align*}
\Omega_{PA} &= \{0/0, +/2, */2, s/1\} \\
\Pi_{PA} &= \{\leq/2, </2\} \\
+, *, <, \leq & \quad \text{infix}; \ast >_p + >_p < >_p \leq
\end{align*}
\]

Examples of formulas over this signature are:

\[
\begin{align*}
\forall x, y(x \leq y & \iff \exists z(x + z \approx y)) \\
\exists x \forall y(x + y \approx y) \\
\forall x, y(x \ast s(y) & \approx x \ast y + x) \\
\forall x, y(s(x) & \approx s(y) \rightarrow x \approx y) \\
\forall x \exists y(x < y & \land \neg \exists z(x < z \land z < y))
\end{align*}
\]