DPLL(T) Rules from DPLL

Unit Propagate:

$$M \parallel N \cup \{C \vee L\} \;\Rightarrow_{\mathrm{DPLL(T)}}\; M\, L \parallel N \cup \{C \vee L\}$$

if C is false under M and L is undefined under M.

Decide:

$$M \parallel N \;\Rightarrow_{\mathrm{DPLL(T)}}\; M\, L^{\mathrm{d}} \parallel N$$

if L is undefined under M.

Fail:

$$M \parallel N \cup \{C\} \;\Rightarrow_{\mathrm{DPLL(T)}}\; \mathit{fail}$$

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

Specific DPLL(T) Rules

T-Backjump:

$$M\, L^{\mathrm{d}}\, M' \parallel N \cup \{C\} \;\Rightarrow_{\mathrm{DPLL(T)}}\; M\, L' \parallel N \cup \{C\}$$

if $M\, L^{\mathrm{d}}\, M' \models \neg C$
there is some "backjump clause" $C' \vee L'$ such that
$N \cup \{C\} \models_T C' \vee L'$ and $M \models \neg C'$
L' is undefined under M', and
L' or $\overline{L'}$ occurs in N or in $M\, L^{\mathrm{d}}\, M'$.

T-Learn:

$$M \parallel N \;\Rightarrow_{\mathrm{DPLL(T)}}\; M \parallel N \cup \{C\}$$

if $N \models_T C$ and each atom of C occurs in N or M.

T-Forget:

$$M \parallel N \cup \{C\} \;\Rightarrow_{\mathrm{DPLL(T)}}\; M \parallel N$$

if $N \models_T C$.

T-Propagate:

$$M \parallel N \;\Rightarrow_{\mathrm{DPLL(T)}}\; M\, L \parallel N$$

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 $\emptyset \parallel N \Rightarrow^*_{\mathrm{DPLL(T)}} M \parallel N'$ 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. $\square$

Lemma 2.3 *If $\emptyset \parallel N \Rightarrow^*_{\mathrm{DPLL(T)}} M \parallel N'$ 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 $\emptyset \models_T \overline{L_1} \vee \ldots \vee \overline{L_n}$. Hence the conflicting clause $\overline{L_1} \vee \ldots \vee \overline{L_n}$ is either in $M \parallel N'$, or else it can be learned by one T-Learn step. $\square$

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)
  $\Rightarrow$ terms, atomic formulas
- logical symbols (domain-independent)
  $\Rightarrow$ Boolean combinations, quantifiers

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 $\mathsf{arity}(p) = m$.

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 Σ (resp., Σ-terms) are formed according to these syntactic rules:

$$\begin{array}{llll} s,t,u,v & ::= & x & , x \in X & \text{(variable)} \\ & \mid f(s_1,...,s_n) & & , f \in \Omega, \mathsf{arity}(f) = n & \text{(functional term)} \end{array}$$

By $\mathrm{T}_\Sigma(X)$ we denote the set of Σ-terms (over X). A term not containing any variable is called a *ground term*. By T_Σ we denote the set of Σ-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 Σ are formed according to this syntax:

$$\begin{array}{llll} A,B & ::= & p(s_1,...,s_m) & , p \in \Pi, \mathsf{arity}(p) = m \\ \Big[& \mid & (s \approx t) & \text{(equation)} \qquad \Big] \end{array}$$

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

$$
\begin{array}{lcll}
L & ::= & A & \text{(positive literal)} \\
 & | & \neg A & \text{(negative literal)}
\end{array}
$$

Clauses

$$
\begin{array}{lcll}
C, D & ::= & \bot & \text{(empty clause)} \\
 & | & L_1 \vee \ldots \vee L_k, \;\; k \geq 1 & \text{(non-empty clause)}
\end{array}
$$

General First-Order Formulas

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

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

Positions in terms, formulas

Positions of a term s (formula F):

$\mathrm{pos}(x) = \{\varepsilon\}$,
$\mathrm{pos}(f(s_1, \ldots, s_n)) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{\, ip \mid p \in \mathrm{pos}(s_i) \,\}$.

$\mathrm{pos}(\forall x F) = \{\varepsilon\} \cup \{\, 1p \mid p \in \mathrm{pos}(F) \,\}$
Analogously for all other formulas.

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

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 $p \in \mathrm{pos}(s)$:

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

Analougously for formulas (F/p).

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

$$s[t]_\varepsilon = t,$$
$$f(s_1, \dots, s_n)[t]_{ip} = f(s_1, \dots, s_i[t]_p, \dots, s_n).$$

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:

- $\neg \quad >_p \quad \vee \quad >_p \quad \wedge \quad >_p \quad \rightarrow \quad >_p \quad \leftrightarrow$
  (binding precedences)
- $\vee$ and $\wedge$ are associative and commutative
- $\rightarrow$ is right-associative

$Qx_1, \dots, x_n\, F$ abbreviates $Qx_1 \dots Qx_n\, F$.

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

Examples:

$s + t * u$	for	$+(s, *(t, u))$
$s * u \leq t + v$	for	$\leq (*(s, u), +(t, v))$
$-s$	for	$-(s)$
0	for	$0()$

Example: Peano Arithmetic

$$\begin{aligned}
\Sigma_{PA} &= (\Omega_{PA},\ \Pi_{PA}) \\
\Omega_{PA} &= \{0/0,\ +/2,\ */2,\ s/1\} \\
\Pi_{PA} &= \{\leq/2,\ </2\}
\end{aligned}$$

$+, *, <, \leq$ infix; $* \ >_p\ + \ >_p\ < \ >_p\ \leq$

Examples of formulas over this signature are:

$\forall x, y(x \leq y \leftrightarrow \exists z(x + z \approx y))$
$\exists x \forall y(x + y \approx y)$
$\forall x, y(x * s(y) \approx x * y + x)$
$\forall x, y(s(x) \approx s(y) \rightarrow x \approx y)$
$\forall x \exists y(x < y \wedge \neg \exists z(x < z \wedge z < y))$