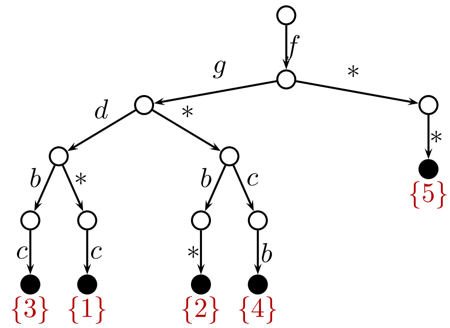

Advantages:

Each leaf yields one term, hence retrieval does not require intersections of intermediate results for subterms.

Good for finding generalizations.

Disadvantages:

Uses more storage than path indexing (due to less sharing).

Uses still more storage, if jump lists are maintained to speed up the search for instances or unifiable terms.

Backtracking required for retrieval.

Literature

Literature:

R. Sekar, I. V. Ramakrishnan, and Andrei Voronkov: Term Indexing, Ch. 26 in Robinson and Voronkov (eds.), *Handbook of Automated Reasoning, Vol. II*, Elsevier, 2001.

Christoph Weidenbach: Combining Superposition, Sorts and Splitting, Ch. 27 in Robinson and Voronkov (eds.), *Handbook of Automated Reasoning, Vol. II*, Elsevier, 2001.

6 Many-Sorted First-Order Logic

Many-Sorted First-order logic

- generalization of first-order logic
- idea is to prohibit ill-defined statements, e.g., $\text{cons}(3, \text{nil}) + 2$
- identical proof theory
- sorts denote subsets of the domain

- variables come with a sort
- functions are declared over the sorts

Many-Sorted Signature

A signature

$$\Sigma_\Upsilon = (\Omega, \Pi, \Upsilon, \upsilon)$$

fixes an alphabet of non-logical symbols, where

- Ω, Π are the sets of function, predicate symbols
- Υ is a set of sort symbols
- υ is a function assigning sorts to function, predicate and variable symbols

Terms, Atoms,Formulae

Well-sorted Terms of sort $S \in \Upsilon$ over Σ_Υ (resp., $\mathrm{T}^S_{\Sigma_\Upsilon}(X)$-terms) are formed according to these syntactic rules:

$$\begin{array}{rcll} s, t, u, v & ::= & x & , x \in X, \upsilon(x) = S \text{ (variable)} \\ & \mid & f(t_1, ..., t_n) & , f \in \Omega, \mathsf{arity}(f) = n, \upsilon(f) = T_1 \ldots T_n S, \\ & & & t_i \in \mathrm{T}^{T_i}_{\Sigma_\Upsilon}(X) \text{ (functional term)} \end{array}$$

By $\mathrm{T}^S_{\Sigma_\Upsilon}$ we denote the set of Σ_Υ-ground terms of sort S, $\mathrm{T}_{\Sigma_\Upsilon}(X) = \bigcup_{S\in\Upsilon} \mathrm{T}^S_{\Sigma_\Upsilon}(X)$.

If $P \in \Pi$, $t_i \in \mathrm{T}^{T_i}_{\Sigma_\Upsilon}(X)$, $\upsilon(P) = T_1 \ldots T_n$ then $P(t_1, ..., t_n)$ is an atom. For any $t, s \in \mathrm{T}^S_{\Sigma_\Upsilon}(X)$, $s \approx t$ is an atom.

Formulae are build as for standard (unsorted) first-order logic.

For substitions we additionally require that if $x\sigma = t$ then $t \in \mathrm{T}^{\upsilon(x)}_{\Sigma_\Upsilon}(X)$ and call it *well-sorted*. Note that application of the standard unification algorithms to any two terms of the same sort yields a well-sorted unifier (if there exists a unifier at all).

Many-Sorted Structures

A Σ_Υ-*algebra* is a quadruple

$$\begin{array}{rl} \mathcal{A} = & (U_\mathcal{A},\ (f_\mathcal{A} : (T_1)_\mathcal{A} \times \ldots \times (T_n)_\mathcal{A} \to S_\mathcal{A})_{f\in\Omega}, \\ & \quad (p_\mathcal{A} \subseteq (S_1)_\mathcal{A} \times \ldots \times (S_m)_\mathcal{A})_{p\in\Pi}, \\ & \quad (T_\mathcal{A} \subseteq U_\mathcal{A})_{T\in\Upsilon}) \end{array}$$

where $\mathsf{arity}(f) = n$, $\mathsf{arity}(p) = m$, $\upsilon(f) = T_1 \ldots T_n S$, $\upsilon(p) = S_1 \ldots S_m$, $T_\mathcal{A} \neq \emptyset$, $U_\mathcal{A} \neq \emptyset$ is a set, called the *universe* of $\mathcal{A}$.

The rest of the semantics is identical to the unsorted case, except that valuations respect the sort information.

7 SUP(LA)

Superposition Modulo Linear Arithmetic

- Consider the base specification $\text{SP} = (\Sigma_{\text{LA}}, \mathcal{A}_{\text{LA}})$, where $\Sigma_{\text{LA}} = (\mathbb{Q} \cup \{+, -, *\}, \{\geq, \leq, >, <\})$ see Section 2.
- The hierarchic extension of SP is $\text{SP}' = (\Sigma', N')$, where $\Sigma_{\text{LA}} \subseteq \Sigma'$ and N' is a set of Σ' clauses.
- We consider a many-sorted setting, consisting of a base sort, containing all terms of Σ_{LA} plus potentially extension terms from $\Sigma' \setminus \Sigma_{\text{LA}}$, and a general sort containing all other terms.
- A term (a clause) consisting only of Σ_{LA} symbols and base sort variables, is called a *base term* (*base clause*).
- For the following results, we need that $\mathcal{A}_{\text{LA}}$ is *term-generated*, i.e., for any $a \in U_{\text{LA}}$ $(= \mathbb{Q})$ there is a ground term $t \in \text{T}_{\Sigma_{\text{LA}}}$ with $\mathcal{A}_{\text{LA}}(t) = a$. This is obvious, because $\mathbb{Q} \subseteq \Sigma_{\text{LA}}$.
- Furthermore, we need that $\text{SP} = (\Sigma_{\text{LA}}, \mathcal{A}_{\text{LA}})$ is compact.
- A model of $\mathcal{A}'$ of SP', i.e., $\mathcal{A}' \models N'$, is called *hierarchic* if $\mathcal{A}' |_{\Sigma_{\text{LA}}} = \mathcal{A}_{\text{LA}}$.
- A substitution is called *simple* if it maps variables of the base sort to base terms.

Hierarchic Clauses

A clause $C = \Lambda \parallel C'$ is called *hierarchic* if Λ only contains base terms and base literals (Σ_{LA}) and all base terms in C' are variables. The semantics of C is $\bigwedge \Lambda \rightarrow C'$.

Any clause can be equivalently transformed into a hierarchic clause: whenever a subterm t whose top symbol is a base theory symbol occurs immediately below a non-base operator symbol, it is replaced by a new base sort variable x ("abstracted out") and the equation $x \approx t$ is added to Λ. Analogously, if a subterm t whose top symbol is not a base theory symbol occurs immediately below a base operator symbol, it is replaced by a general variable y and the disequation $y \not\approx t$ is added to C'. This transformation is repeated until the clause is hierarchic.

Superposition Modulo LA

Pos. Superposition:

$$\frac{\Lambda_1 \parallel D' \lor t \approx t' \qquad \Lambda_2 \parallel C' \lor s[u] \approx s'}{(\Lambda_1, \Lambda_2 \parallel D' \lor C' \lor s[t'] \approx s')\sigma}$$

where $\sigma = \mathrm{mgu}(t, u)$ and simple and u is not a variable.

Neg. Superposition:

$$\frac{\Lambda_1 \parallel D' \lor t \approx t' \qquad \Lambda_2 \parallel C' \lor s[u] \not\approx s'}{(\Lambda_1, \Lambda_2 \parallel D' \lor C' \lor s[t'] \not\approx s')\sigma}$$

where $\sigma = \mathrm{mgu}(t, u)$ and simple and u is not a variable.

Equality Resolution:

$$\frac{\Lambda \parallel C' \lor s \not\approx s'}{(\Lambda \parallel C')\sigma}$$

where $\sigma = \mathrm{mgu}(s, s')$ and simple.

Equality Factoring:

$$\frac{\Lambda \parallel C' \lor s' \approx t' \lor s \approx t}{(\Lambda \parallel C' \lor t \not\approx t' \lor s \approx t')\sigma}$$

where $\sigma = \mathrm{mgu}(s, s')$ and simple.

Constraint Refutation:

$$\frac{\Lambda \parallel \Box}{\Box}$$

where $\neg(\bigwedge \Lambda)$ is inconsistent in $\mathcal{A}_{\mathrm{LA}}$.

Redundancy

A clause $C \in N$ is called *redundant* if for all simple ground instances C' of C there are simple ground instances $C'_1, \ldots, C'_n$ from N such that $C'_1, \ldots, C'_n \models C'$ and $C'_i \prec C'$ for all i.

A hierarchic clause $\Lambda \parallel C$ is called a *tautology* if C is a tautology or the existential closure of $\bigwedge \Lambda$ is unsatisfiable in $\mathcal{A}_{\mathrm{LA}}$.

A hierarchic clause $\Lambda_1 \parallel C_1$ *subsumes* a hierarchic clause $\Lambda_2 \parallel C_2$, if there is a simple matcher σ such that $C_1\sigma \subset C_2$ and the universal closure of $\bigwedge \Lambda_2 \rightarrow \bigwedge \Lambda_1\sigma$ holds in $\mathcal{A}_{\mathrm{LA}}$.

Purely base sort variable equations generated during reasoning are moved from the FOL to the LA part.

Sufficient Completeness

A set N of clauses is called *sufficiently complete with respect to simple instances*, if for every model $\mathcal{A}'$ of the set of simple ground instances from N and every ground non-base term t of the base sort there exists a ground base term t such that $t' \approx t$ is true in $\mathcal{A}'$.

Completeness of SUP(LA)

The hierarchic superposition calculus modulo LA is refutationally complete for all sets of clauses that are sufficiently complete with respect to simple instances.

Current Hot Research Topics & Applications

- decidability of SUP(LA) $\Rightarrow$ automata theory, software analysis
- better/different calculi for SAT $\Rightarrow$ configuration management
- parallel calculi for SAT/FOF $\Rightarrow$ graphics hardware
- scalable calculi for Finite Domain FOF $\Rightarrow$ knowledge management
- understanding the combination of FOF with theories $\Rightarrow$ insight

The End