Literature

Leo Bachmair, Harald Ganzinger: Rewrite techniques for transitive relations. IEEE Symposium on Logic in Computer Science, LICS-9, pp. 384–393, 1994.

Leo Bachmair, Harald Ganzinger: Ordered chaining for total orderings. Automated Deduction, CADE-12, LNAI 814, pp. 435–450, Springer, 1994.

Leo Bachmair, Harald Ganzinger: Associative-commutative superposition. Conditional and Typed Rewriting Systems, CTRS-94, LNCS 968, pp. 1–14, Springer, 1994.

E. Cardoza, R. Lipton, A. R. Meyer: Exponential space complete problems for Petri nets and commutative semigroups: preliminary report. Eighth Annual ACM Symposium on Theory of Computing, STOC, pp. 50–54, 1976.

Guillem Godoy, Robert Nieuwenhuis: Paramodulation with built-in Abelian groups. IEEE Symposium on Logic in Computer Science, LICS-15, pp. 413–424, 2000.

Ernst W. Mayr, Albert R. Meyer: The complexity of the word problems for commutative semigroups and polynomial ideals. Advances in Mathematics, 46(3):305–329, 1982.

Gerald E. Peterson, Mark E. Stickel: Complete sets of reductions for some equational theories. Journal of the ACM, 28(2):233–264, 1981.

Uwe Waldmann: Cancellative abelian monoids and related structures in refutational theorem proving (Part I & II). Journal of Symbolic Computation, 33(6):777–829/831–861, 2002.

Uwe Waldmann: Superposition and chaining for totally ordered divisible abelian groups. Technical report MPI-I-2001-2-001, Max-Planck-Institut für Informatik, Saarbrücken, 2001.

Ulrich Wertz: First-order theorem proving modulo equations. Technical report MPI-I-92-216, Max-Planck-Institut für Informatik, Saarbrücken, 1992.

4 Higher-Order Logic

Higher-Order logic

- extends first-order logic with quantification over functions and predicates
- is *very* expressive (natural numbers, uncountable sets...)
- is the preferred language of most mathematicians

Higher-order logic is also called *simple type theory*.

4.1 History

Higher-order quantification

Unrestricted quantification is first considered by Frege (1879).

It contains several paradoxical statements, such as Russell's paradox, which motivated the creation of *ramified type theory* by Russell (1908).

A later simplification of this theory by Church (1940) was denoted a *simple type theory*, or HOL.

4.2 Syntax

Syntax choices:

explicit function symbols

explicit predicate abstraction

quantifiers and connectives as constants of the language

with extensionality

Types

Types are defined recursively:

o is the type of Booleans, of order 0.

ι is the type of individuals, of order 1.

if τ_1 and τ_2 are types then $\tau_1 \to \tau_2$ is a type, of order $\max(\text{order}(\tau_1) + 1, \text{order}(\tau_2))$

We also use the notation $\tau_1, \ldots, \tau_n \to \tau$ to denote $\tau_1 \to (\cdots \to (\tau_n \to \tau) \ldots)$.

Terms

Given a non-empty set of constants and a collection of non-empty sets of variables for each type,

constants are terms

variables are terms

if t_1 and t_2 are terms then $(t_1 t_2)$ is a term

if x is a variable and t is a term then $\lambda x.\, t$ is a term

Types of Terms

Given a non-empty set S of individuals and a collection of non-empty sets of variables for each type, the term t is of type

o if $t \in \{\top, \bot\}$

ι if $t \in S$

τ if $t = x_{(\tau)}$ is a variable of type τ

$\tau_1 \to \tau_2$ if $t = \lambda x_{(\tau_1)}.\, t_{1(\tau_2)}$

τ_2 if $t = (t_{1(\tau_1 \to \tau_2)}\ t_{2(\tau_1)})$

A term is well-typed if a type can be associated to it according to the previous definition. We only consider well-typed terms in what follows.

4.3 Semantics

A well-founded formula is a term of type o.

How to evaluate the truth of such a formula?

Classical Model

Let D be a non-empty set, for each type τ we define the following collection, denoted as the *frame* of the type

the frame of $\tau = o$ is $[\![o, D]\!] = \{\top, \bot\}$

the frame of $\tau = \iota$ is $[\![\iota, D]\!] = D$

the frame of $\tau = \tau_1 \to \tau_2$ is $[\![\tau_1 \to \tau_2, D]\!]$, the collection of all functions mapping $[\![\tau_1, D]\!]$ into $[\![\tau_2, D]\!]$

A higher-order *classical model* is a structure $\mathcal{M} = \langle D, \mathcal{I} \rangle$ where D is a non-empty set called the *domain* of the model and $\mathcal{I}$ is the *interpretation* of the model, a mapping such that

if $a_{(\tau)}$ is a constant then $\mathcal{I}(a) \in [\![\tau, D]\!]$,

$\mathcal{I}(=_{(\tau \to \tau \to o)})$ is the equality relation on $[\![\tau, D]\!]$.

By adding a *valuation* function α such that for any variable $x_{(\tau)}$, $\alpha(x) \in [\![\tau, D]\!]$, it becomes possible to evaluate the truth-value of higher-order formulas as in first-order logic.

The *evaluation* $\mathcal{V}_{\mathcal{M},\alpha}(t)$ of a term t given a model $\mathcal{M} = \langle D, \mathcal{I} \rangle$ and a valuation α is recursively defined as

$\mathcal{I}(a)$ if t is a constant a

$\alpha(x)$ if t is a variable x

the function from $[\![\tau_1, D]\!]$ to $[\![\tau_2, D]\!]$ such that for all $a \in [\![\tau_1, D]\!]$, $(\mathcal{V}_{\mathcal{M},\alpha}(\lambda x.\, t))(a) = \mathcal{V}_{\mathcal{M},\alpha}(t[a/x])$ if $t = \lambda x_{(\tau_1)}.\, t_{(\tau_2)}$

$(\mathcal{V}_{\mathcal{M},\alpha}(t_1))(\mathcal{V}_{\mathcal{M},\alpha}(t_2))$ if $t = (t_{1(\tau_1 \to \tau_2)}\ t_{2(\tau_1)})$

Truth evaluation:

Given a model $\mathcal{M} = \langle D, \mathcal{I} \rangle$ and a valuation α, a well-founded formula ϕ is true in $\mathcal{M}$ with respect to α, denoted as $\mathcal{M}, \alpha \models \phi$ iff $\mathcal{V}_{\mathcal{M},\alpha}(\phi) = \{\top\}$

ϕ is satisfiable in $\mathcal{M}$ iff there exist a valuation α such that $\mathcal{M}, \alpha \models \phi$

ϕ is valid in $\mathcal{M}$, denoted $\mathcal{M} \models \phi$ iff for all valuations α, $\mathcal{M}, \alpha \models \phi$

ϕ is valid, denoted $\models \phi$ iff for all models $\mathcal{M}$, $\mathcal{M} \models \phi$

These notions extend straightforwardly to sets of formulas.

Problems with the classical semantic

- Loss of compactness: in FOL, every unsatisfiable set of formulas has a finite unsatisfiable subset. This is no longer the case in HOL with classical semantics (cHOL).
- Loss of strong completeness: no proof procedure able to derive all consequences of a set of formulas can exist in cHOL.
- Loss of weak completeness: no proof procedure able to derive all valid sets of formulas can exist in cHOL.
- And even worse: the status of validity of some formulas is unclear.

Henkin Semantics

To solve the previously mentioned issues, it is possible to generalize the notion of a model by relaxing the notion of a frame into that of a Henkin frame. Given a non-empty set D,

$[\![o, D]\!] = \{\top, \bot\}$

$[\![\iota, D]\!] = D$

$[\![\tau_1 \to \tau_2, D]\!]$ is the collection of ~~all~~ *some* functions mapping $[\![\tau_1, D]\!]$ into $[\![\tau_2, D]\!]$ *with some additional closure conditions*.

Henkin vs Classical Semantics

- Any formula true in all Henkin models is true in all classical models.
- There are formulas true in all classical models that are not true in all Henkin models.
- There are (weak) complete proof procedures for HOL with Henkin semantics.

4.4 Higher-Order Term Unification

In FOL, there exist a unique m.g.u. for two terms.

This is no longer true in HOL.

For example, consider $t_1 = f\ x$ and $t_2 = a$ where f, x are variables and a is a constant. The unifiers of t_1 and t_2 are $\{f \mapsto \lambda y.\, a\}$ and $\{f \mapsto \lambda y.\, y, x \mapsto a\}$.

Some equations even have an infinite number of m.g.u's.

Even worse, the higher-order unification problem is *undecidable*.

Huet's Unification Algorithm

Given:

E, a unification problem, i.e. a finite set of equations.

Goal:

find a substitution σ such that $E\sigma$ contains only syntactically equal equations.

Idea:

Test if the head symbols of the two sides of equations can be unified or not to restrict the search space.

Rigid and Flexible Terms

A term is *rigid* if its head symbol is a constant or a bound variable. Otherwise its head symbol is a free variable and the term is *flexible*.

Rigid-Rigid Equations

Two rules can be applied depending on the head symbols in the rigid-rigid equation.

Fail:

$$\frac{E \cup \{\lambda x_1 \ldots x_n.\, f\ u_1 \ldots u_p \approx \lambda x_1 \ldots x_n.\, g\ v_1 \ldots v_q\}}{\bot}$$

Simplify:

$$\frac{E \cup \{\lambda x_1 \ldots x_n.\, f\ u_1 \ldots u_p \approx \lambda x_1 \ldots x_n.\, f\ v_1 \ldots v_p\}}{E \cup \{\lambda x_1 \ldots x_n.\, u_1 \approx \lambda x_1 \ldots x_n.\, v_1, \ldots, \lambda x_1 \ldots x_n.\, u_p \approx \lambda x_1 \ldots x_n.\, v_p\}}$$

Flexible-Rigid Equations

There is only one rule to handle such terms, but it can generate many results.

Generate:

$$\frac{E}{E\sigma}$$

where $\sigma = \{X \mapsto \lambda y_1, \ldots, y_p.\, h\ (H_1\ y_1 \ldots y_p) \ldots (H_r\ y_1 \ldots y_p)\}$ and $(\lambda x_1 \ldots x_n.\, X\ u_1 \ldots u_p \approx \lambda x_1 \ldots x_n.\, f\ \ v_1 \ldots v_q) \in E$ such that $h \in \{f, y_1, \ldots, y_p\}$ if f is a constant and $h \in \{y_1, \ldots, y_p\}$ otherwise.

Flexible-Flexible Equations

The following result, also by Huet, handles flexible-flexible equations.

Proposition 4.1 *A unification problem E containing only flexible-flexible equations has always a solution.*

Proof. Consider any flexible-flexible equation

$$e = (\lambda x_1 \dots x_n.\, X_{(\tau)}\; u_1 \dots u_p \approx \lambda x_1 \dots x_n.\, Y\; v_1 \dots v_q).$$

Since there are no empty types, there exists a constant $a_{(\tau)}$ for each type τ. Let θ_τ be the substitution that maps all variables of type τ to this constant a. Then $e\theta = (\lambda x_1 \dots x_n.\, a \approx \lambda x_1, \dots, x_n.\, a)$ thus θ is a unifier of e.

We say that a unification problem with only flexible-flexible equations is a *solved* unification problem.

The Whole Procedure

A reasonable strategy consists in applying Fail and Simplify eagerly, and Generate only when there is no rigid-rigid equation left.

Generate is non-deterministic, making this procedure branching.

Theorem 4.2 *The procedure made of the rules Fail, Simplify and Generate is sound and complete.*

Soundness

Proposition 4.3 *If a unification problem E can be transformed into a solved problem E' by applying Fail, Simplify and Generate then E has a solution.*

The proof is by induction on the size of the derivation from E to E'.

- If E is a solved problem then Prop. 4.1 applies.
- If the first rule applied on E is Fail then $E' = \perp$ is not a solved problem, a contradiction.

- If the first rule applied on E is Simplify, resulting in E_1, then by the induction hypothesis, E_1 has a solution σ and

$$E = E_0 \cup \{\lambda x_1 \dots x_n.\, f\ u_1 \dots u_p \approx \lambda x_1 \dots x_n.\, f\ v_1 \dots v_p\}$$
$$\text{and } E_1 = E_0 \cup \{\lambda x_1 \dots x_n.\, u_1 \approx \lambda x_1 \dots x_n.\, v_1, \dots, \lambda x_1 \dots x_n.\, u_p \approx \lambda x_1 \dots x_n.\, v_p\}$$

  such that $\sigma(u_i) = \sigma(v_i)$ for all $i \in \{1, \dots, p\}$ and $E_0\sigma$ contains only trivial equations. Thus $E\sigma = E_0\sigma \cup \{(\lambda x_1 \dots x_n.\, f\ u_1 \dots u_p \approx \lambda x_1 \dots x_n.\, f\ v_1 \dots v_p)\sigma\}$. Since $(f\ u_1 \dots u_p)\sigma = f\ u_1\sigma \dots u_p\sigma = f\ v_1\sigma \dots v_p\sigma = (f\ v_1 \dots v_p)\sigma$, the substitution σ is a solution to the unification problem E.

- If the first rule applied on E is Generate and E_1 is the resulting unification problem then there exists a solution σ to E_1 by the induction hypothesis, and $E_1 = E\theta$ where $\theta = \{X \mapsto \lambda y_1 \dots y_p.\, h\ (H_1\ y_1 \dots y_p) \dots (H_r\ y_1 \dots y_p)\}$ and E contains an equation $\lambda x_1 \dots x_n.\, X\ u_1 \dots u_p \approx \lambda x_1 \dots x_n.\, f\ v_1 \dots v_q$ Let $\sigma' = \sigma \circ \theta$. Since $E\sigma' = E(\sigma \circ \theta) = (E\theta)\sigma = E_1\sigma$, the substitution σ' is a solution of E.

Completeness

Proposition 4.4 *If a unification problem E has a solution σ then we can derive a solved problem E' from E using the rules Fail, Simplify and Generate.*

Proof. This proof is done by induction of the complexity of σ. First we must define this measure. Let us consider an arbitrary term $t = \lambda x_1 \dots x_n.\, h\ u_1 \dots u_p$. The number of applications used in t, denoted $\pi(t)$ is computed in the following way: $\pi(t) = p + \sum_{i=1}^{p} \pi(u_i)$. This function is used to compute the complexity of a substitution. Let σ be a substitution that maps the variables x_1,...,x_k to the terms t_1,...,t_k such that x_i and t_i are distinct, and that maps all other variables to themselves. The complexity $\mathcal{C}(\sigma)$ of σ is $\mathcal{C}(\sigma) = k + \sum_{i=1}^{k} \pi(t_i)$. We can now use this measure to perform an induction in the following way.

- If E contains rigid-rigid equations then it is possible to get rid of them by repeatedly applying the rule Simplify. This process terminates, which can be proved by induction on the size of terms in E. Fail can never be applied during this process, otherwise it would contradict the fact that there exist a solution of E. If the resulting set of equations E_s is solved, we have a derivation to a solved problem.

- Otherwise E_s contains no rigid-rigid equations, but at least one flexible-rigid one, i.e., $\lambda x_1 \dots x_n.\, X\ u_1 \dots u_p \approx \lambda x_1 \dots x_n.\, f\ v_1 \dots v_q \in E_s$. Since σ is a solution of E_s, $(\lambda x_1 \dots x_n.\, X\ u_1 \dots u_p)\sigma = (\lambda x_1 \dots x_n.\, f\ v_1 \dots v_q)\sigma$, thus $\sigma = \sigma_0 \cup \theta$ where $\theta = \{X \mapsto \lambda y_1 \dots y_p.\, h\, w_1 \dots w_r\}$, and $h \in \{y_1, \dots, y_p, f\}$ if f is a constant or $h \in \{y_1, \dots, y_p\}$ otherwise. Then, we can use Generate with the function h occurring in θ on E_s to generate $E_s' = E_s\theta'$ where $\theta' = X \mapsto \lambda y_1, \dots, y_p.\, h\ (H_1\, y_1 \dots y_p) \dots$

$(H_r\ y_1 \dots y_p)$. Let $\gamma = \{H_1 \mapsto \lambda y_1 \dots y_p. w_1, \dots, H_r \mapsto \lambda y_1 \dots y_p. w_r\}$. The complexity of σ is $\mathcal{C}(\sigma) = \mathcal{C}(\sigma_0 \cup \theta) = \mathcal{C}(\sigma_0) + 1 + r + \sum_{i=1}^{r} \pi(w_i)$, and that of σ' is $\mathcal{C}(\sigma') = \mathcal{C}(\sigma_0 \cup \gamma) = \mathcal{C}(\sigma_0) + r + \sum_{i=1}^{r} \pi(w_i)$. Thus $\mathcal{C}(\sigma') < \mathcal{C}(\sigma)$. By the induction hypothesis, there exists a derivation from E'_s to a solved unification problem E' and E'_s was obtained by derivation from E hence we can conclude.

Termination?

Higher-order unification is only semi-decidable.

When solutions exist, Huet's algorithm will find one and terminate, but when there is no solution, it may loop forever.

4.5 Resolution in Higher-Order Logic

In first-order logic, resolution for general clauses has two rules:

Resolution: $$\frac{D \vee B \qquad C \vee \neg A}{(D \vee C)\sigma}$$
where $\sigma = \text{mgu}(A, B)$.

Factoring: $$\frac{C \vee A \vee B}{(C \vee A)\sigma}$$
where $\sigma = \text{mgu}(A, B)$.

In higher-order logic, a first problem is that m.g.u's need not exist and unification is undecidable.

Example 4.5 *Given $D \vee B$ and $C \vee \neg A$ where A and B are unifiable but without m.g.u., there may exist infinitely many σ_1, σ_2,... unifiers of A and B generating distinc resolvents $(D \vee C)\sigma_1$, $(D \vee C)\sigma_2$,... and in general there is no way to know which one is needed to prove the given theorem.*

Huet proposes to delay the computation of unifiers (when no m.g.u. exists) by using constraints storing the corresponding unification problems.

Once a contradiction has been derived, the corresponding unification problem can then be solved using Huet's algorithm.

Resolution: $$\frac{D \vee B[\![X]\!] \qquad C \vee \neg A[\![Y]\!]}{D \vee C[\![X \wedge Y \wedge A \approx B]\!]}$$

Factoring: $$\frac{C \vee A \vee B[\![X]\!]}{C \vee A[\![X \wedge A \approx B]\!]}$$

Another problem in HOL is that it is not always possible to guess the necessary substitution based on the available terms.

Example 4.6 *Consider the formula $\neg X_{(o)}$ where X is a Boolean variable. The set $\{\neg X\}$ is saturated by resolution, but still the formula $\neg X$ is unsatisfiable. However, we can guess the substitution $\sigma = \{X \mapsto \neg Y\}$. Then $(\neg X)\sigma = \neg(\neg Y) = Y$ and resolution can now derive the empty clause from $\neg X$ and Y.*

To overcome this issue, Huet introduces additional *splitting rules*.

$$\frac{C \vee A[\![X]\!]}{C \vee \neg x_{(o)}[\![X \wedge A \approx \neg x]\!]}$$

$$\frac{C \vee A[\![X]\!]}{C \vee x_{(o)} \vee y_{(o)}[\![X \wedge A \approx (x \vee y)]\!]}$$

$$\frac{C \vee A[\![X]\!]}{C \vee P_{(\tau\to o)}x_{(\tau)}[\![X \wedge A \approx \Pi_{((\tau\to o)\to o)}P]\!]}$$

$\Pi_{((\tau\to o)\to o)}$ is the function that associates $\top$ to any set of type $\tau \to o$ that contains all elements of type τ.

$$\frac{C \vee \neg A[\![X]\!]}{C \vee x_{(o)}[\![X \wedge A \approx \neg x]\!]}$$

$$\frac{C \vee \neg A[\![X]\!]}{C \vee x_{(o)}[\![X \wedge A \approx (x \vee y_{(o)})]\!] \text{ and } C \vee y[\![X \wedge A \approx (x \vee y)]\!]}$$

$$\frac{C \vee \neg A[\![X]\!]}{C \vee \neg P_{(\tau\to o)}(\mathrm{sk}_{((\tau\to o)\to\tau)}P)[\![X \wedge A \approx \Pi_{((\tau\to o)\to o)}P]\!]}$$

sk is the skolem constant such that $\neg\Pi_{((\tau\to o)\to o)}P = \neg P_{(\tau\to o)}(\mathrm{sk}_{((\tau\to o)\to\tau)}P)$.

Huet proved that resolution with these splitting rules is sound and complete (but not terminating).

In practice, several improvements are possible.

As soon as a constraint becomes unsatisfiable, delete the corresponding clause.

If a constraint has a small enough set of solutions, generate all applied clauses to replace the constrained original one.

4.6 Superposition in Higher-Order Logic

In HOL, existing automated solvers rely on:

- Tableau (Satallax)
- Resolution (Leo III)
- Applicative encoding to first-order logic (Sledgehammer)
- ...

Currently there exists no efficient version of Superposition for full higher-order logic.

There are many theoretical problems to lifting Superposition to HOL (unification,...)

Superposition in λ-free Higher-Order Logic

Things get easier in λ-free higher-order logic (i.e. no λ-terms and no predicate variables).

This fragment can be encoded in FOL using a binary function *app* (application).

If the ordering has all standard properties of reduction orderings plus compatibility with arguments, the extension of Superposition to this fragment is straightforward.

There is only one known ordering with these properties: KBO.

There are applications where standard KBO is not optimal.

There are other orderings that one would like to use (LPO, KBO with multipliers) but one loses at least one of the desired properties (e.g. compatibility with arguments).

There are workarounds that allow to recover from the loss of compatibility with arguments, e.g. by:

redefining redundancy so that $g\ x \approx f\ x$ is not redundant to $g \approx f$, and

adding a rule that adds context to an equation (generate $g\ x \approx f\ x$ from $g \approx f$), and

relaxing the variable constraint in the superposition rules (no superposition at or under a variable, except if...), and

adding a layer to the completeness proof.

Our current goal is to extend this calculus to predicate-free HOL (including λ-terms) and then to full HOL.

Literature

Peter B. Andrews: An introduction to mathematical logic and type theory - to truth through proof. Computer science and applied mathematics, Academic Press, ISBN 978-0-12-058535-9, pp. I-XV, 1-304, 1986.

Peter B. Andrews: Classical Type Theory. Handbook of Automated Reasoning: 965-1007, 2001.

Heiko Becker, Jasmin Christian Blanchette, Uwe Waldmann, Daniel Wand: A Transfinite Knuth-Bendix Order for Lambda-Free Higher-Order Terms. CADE: 432-453, 2017.

Alexander Bentkamp, Jasmin Christian Blanchette, Simon Cruanes, and Uwe Waldmann: Superposition for lambda-free higher-order logic. IJCAR: (to appear), 2018.

Christoph Benzmüller, Dale Miller: Automation of Higher-Order Logic. Computational Logic: 215-254, 2014.

Jasmin Christian Blanchette, Uwe Waldmann, Daniel Wand: A Lambda-Free Higher-Order Recursive Path Order. FoSSaCS: 461-479, 2017.

Gilles Dowek: Higher-Order Unification and Matching. Handbook of Automated Reasoning: 1009-1062, 2001.

Melvin Fitting: Types Tableaus and Gödel's God. Studia Logica 81(3): 425-427, 2005.

Gérard P. Huet: A Mechanization of Type Theory. IJCAI: 139-146, 1973.