An Abstract HOL-Kernel

Steven Obua *

Technische Universität München
D-85748 Garching, Boltzmannstr. 3, Germany

An abstract HOL-kernel knows the primitive datatypes that are described in Figure 1. In Figure 2 we define the free *types* and *terms* of the kernel inductively

$x \in \texttt{nat}$	x is a non-negative integer
$x \in \texttt{string}$	x is a string
$x \in \texttt{id}$	x is a string
$x \in \texttt{tag}$	x is a list of strings

Fig. 1. The primitive datatypes of an abstract HOL kernel

as sets `type` and `term`, respectively. The set of *variables* `var` and the set of *constants* `const` are subsets of `term`; the set of *type variables* `tvar` is a subset of `type`. Equality on elements of `type` and `term` is defined as usual by comparing the constructors and, if necessary, their arguments. Therefore $f \cdot g$ and $\lambda x.\, b$ are obviously not equal; and we have $\lambda x.\, b = \lambda x'.\, b'$ iff both $x = x'$ and $b = b'$ hold.

$$\frac{\alpha \in \texttt{string}}{\texttt{TVar}\ \alpha \in \texttt{tvar}} \qquad \frac{\alpha \in \texttt{tvar}}{\alpha \in \texttt{type}} \qquad \frac{c \in \texttt{id} \quad T_1 \in \texttt{type} \quad \cdots \quad T_n \in \texttt{type}}{\texttt{TApp}\,(c, T_1, \ldots, T_n) \in \texttt{type}}$$

$$\frac{x \in \texttt{string} \quad T \in \texttt{type}}{\texttt{Var}\,(x, T) \in \texttt{var}} \qquad \frac{x \in \texttt{var}}{x \in \texttt{term}} \qquad \frac{c \in \texttt{id} \quad T \in \texttt{type}}{\texttt{Const}\,(c, T) \in \texttt{const}} \qquad \frac{c \in \texttt{const}}{c \in \texttt{term}}$$

$$\frac{f \in \texttt{term} \quad g \in \texttt{term}}{f \cdot g \in \texttt{term}} \qquad \frac{x \in \texttt{var} \quad b \in \texttt{term}}{\lambda x.\, b \in \texttt{term}}$$

Fig. 2. Free types and terms

A kernel state S consists of

– the finite partial map $\mathit{arity}_S \subseteq \texttt{id} \times \texttt{nat}$

* Supported by the Ph.D. program "Logik in der Informatik" of the "Deutsche Forschungsgemeinschaft."

– and the finite partial map $ctype_S \subseteq \texttt{id} \times \texttt{type}$.

An HOL-kernel comes with certain *reserved names* for type and term constants. These names are elements of `id` and fixed for a given concrete kernel; but they are allowed to vary among different concrete kernels. We denote the two reserved type names by *fun* and *bool*. For each kernel state S we demand $arity_S(fun) = 2$ and $arity_S(bool) = 0$. The type of a term t is denoted $\tau(t)$ and defined as a partial map from `term` to `type` (fig. 3). In the rules for τ and in what follows we employ

$$\tau(\texttt{Var}\,(x,T)) = T \qquad \tau(\texttt{Const}\,(c,T)) = T$$

$$\frac{\tau(f) = A \to B \qquad \tau(g) = A}{\tau(f \cdot g) = B} \qquad \tau(\lambda\, x.\, b) = \tau(x) \to \tau(b)$$

Fig. 3. Type $\tau(t)$ of a term t

the abbreviations $A \to B \;:=\; \texttt{TApp}\,(fun, A, B)$ and $\texttt{bool} := \texttt{TApp}(bool)$. Note that as a consequence of the definition of τ, τ is not smart enough to perform matching in order to become less partial; for instance if $\tau(f) = \texttt{TVar}\,\alpha \to \texttt{bool}$ and $\tau(g) = \texttt{bool}$, then $\tau(f \cdot g)$ is not defined.

For a given kernel state S, certain types and terms make sense and some don't. We call those which do make sense *well-formed with respect to* S and gather them into the sets $\texttt{type}_S$ and $\texttt{term}_S$ as defined in Figure 4. Special subsets

$$\frac{\alpha \in \texttt{tvar}}{\alpha \in \texttt{type}_S}\ (\textsc{TVar}) \qquad \frac{x \in \texttt{string} \qquad T \in \texttt{type}_S}{\texttt{Var}\,(x,T) \in \texttt{term}_S}\ (\textsc{Var})$$

$$\frac{c \in \texttt{id} \qquad T_1 \in \texttt{type}_S \qquad \cdots \qquad T_n \in \texttt{type}_S \qquad arity_S(c) = n}{\texttt{TApp}\,(c, T_1, \ldots, T_n) \in \texttt{type}_S}\ (\textsc{TApp})$$

$$\frac{c \in \texttt{id} \qquad T \in \texttt{type}_S \qquad defined_S(c, T') \qquad T = \sigma\, T'}{\texttt{Const}\,(c,T) \in \texttt{term}_S}\ (\textsc{Const})$$

$$\frac{f \in \texttt{term}_S \qquad g \in \texttt{term}_S \qquad \tau(f) = A \to B \qquad \tau(g) = A}{f \cdot g \in \texttt{term}_S}\ (\textsc{App})$$

$$\frac{\lambda\, x.\, b \in \texttt{term} \qquad \tau(x) \in \texttt{type}_S \qquad b \in \texttt{term}_S}{\lambda\, x.\, b \in \texttt{term}_S}\ (\textsc{Abs})$$

Fig. 4. Well-formed types and terms with respect to a kernel state S

of $\texttt{term}_S$ are the set of well-formed variables $\texttt{var}_S := \texttt{var} \cap \texttt{term}_S$ and the set of well-formed *propositions* $\texttt{prop}_S := \{p \in \texttt{term}_S \mid \tau\,(p) = \texttt{bool}\}$.

In the CONST-rule, σ is a *type substitution*, that is a finite partial map from `tvar` to `type`. There are also *term substitutions*, which are finite partial maps from `var` to `term`. For explicitly writing down a substitution σ we use the notation

$$\sigma = \langle\, \mathit{redex}_1 \mapsto \mathit{residue}_1, \ldots,\ \mathit{redex}_n \mapsto \mathit{residue}_n \,\rangle\,.$$

For type substitutions we have $\mathit{redex}_i \in \texttt{tvar}$, $\mathit{residue}_i \in \texttt{type}$, similarly for term substitutions $\mathit{redex}_i \in \texttt{var}$, $\mathit{residue}_i \in \texttt{term}$. In any case we adopt the convention that $\mathit{redex}_i = \mathit{redex}_j$ iff $i = j$. We call the sets of substitutions that involve only well-formed redexes and residues $\texttt{tsubst}_S$ and $\texttt{subst}_S$, respectively. We reserve the notation $\sigma_{\setminus x}$ for the substitution that arises from σ by removing that redex/residue pair $\mathit{redex} \mapsto \mathit{residue}$ for which $\mathit{redex} = x$ holds, if such a pair exists, and otherwise leaving the substitution unchanged.

A type substitution is extended in the obvious way to a total function on types and terms (fig. 5). Extending a term substitution to a total function on

$$\sigma\,\alpha = \begin{cases} T & \text{if } \alpha \mapsto T \text{ is a redex/residue pair of } \sigma \\ \alpha & \text{if } \alpha \in \texttt{tvar} \text{ and there is no such redex/residue pair} \end{cases}$$

$$\sigma\,(\texttt{TApp}\,(c, T_1, \ldots, T_n)) = \texttt{TApp}\,(c, \sigma\,T_1, \ldots, \sigma\,T_n) \qquad \sigma\,(\texttt{Var}\,(\alpha, T)) = \texttt{Var}\,(\alpha, \sigma\,T)$$

$$\sigma\,(\texttt{Const}\,(c, T)) = \texttt{Const}\,(c, \sigma\,T) \qquad \sigma\,(f \cdot g) = (\sigma\,f) \cdot (\sigma\,g) \qquad \sigma\,(\lambda\,x.\,b) = \lambda\,\sigma\,x.\,\sigma\,b$$

Fig. 5. Applying a type substitution σ

terms is more involved because we have to be aware of name clashes of variables in λ-abstractions. We need the notion of *free variables* of terms and substitutions as defined in Figure 6. Then Figure 7 shows how term substitutions are applied.

$$\frac{x \in \texttt{var}}{\mathrm{frees}\,(x) = \{x\}} \qquad \frac{c \in \texttt{const}}{\mathrm{frees}\,(c) = \emptyset} \qquad \mathrm{frees}\,(f \cdot g) = \mathrm{frees}\,(f, g)$$

$$\mathrm{frees}\,(\lambda\,x.\,b) = \mathrm{frees}\,(b) - x$$

$$\mathrm{frees}\,(t_1, \ldots, t_n) = \mathrm{frees}\,(\{t_1, \ldots, t_n\}) = \mathrm{frees}\,(t_1) \cup \ldots \cup \mathrm{frees}\,(t_n)$$

$$\mathrm{frees}\,(\langle\, x_1 \mapsto t_1, \ldots,\ x_n \mapsto t_n \,\rangle) = \{x_1, \ldots, x_n\} \cup \mathrm{frees}\,(t_1, \ldots, t_n)$$

Fig. 6. Free variables of terms and term substitutions

Obviously the result r of applying a substitution σ to a term t depends on the choices made for y in the λ-case. Therefore r is not a term, but rather a set of terms, the set of all possible outcomes. Of course we will forget about this fact and just pick one of the possible outcomes, but for the next definition only let us write $\mathit{Outcomes}\,(\sigma, t)$ for this set. Thus we say that two terms s and t are *α-equivalent* iff applying the empty substitution to both terms yields the same set of outcomes:

$$s \stackrel{\alpha}{=} t \quad \text{iff} \quad \mathit{Outcomes}\,(\langle\,\rangle, s) = \mathit{Outcomes}\,(\langle\,\rangle, t).$$

The abstract HOL-kernel and all the concrete ones always operate modulo at

$$\frac{c \in \texttt{const}}{\sigma\, c = c} \qquad \sigma\, x = \begin{cases} t & \text{if } x \mapsto t \text{ is a redex/residue pair of } \sigma \\ x & \text{if } x \in \texttt{var} \text{ and there is no such redex/residue pair} \end{cases}$$

$$\sigma\,(f \cdot g) = (\sigma\, f) \cdot (\sigma\, g)$$

$$\sigma\,(\lambda\, x.\, b) = \lambda\, y.\, \sigma_{\setminus x}\,(\langle\, x \mapsto y\,\rangle\; b), \text{ where } y \in \texttt{var},\ \tau\,(x) = \tau\,(y) \text{ and } y \notin \mathrm{frees}\,(\sigma_{\setminus x}, b)$$

Fig. 7. Applying a term substitution σ

least α-equivalence on terms, therefore there is no harm in considering $\sigma\, t$ a term if t is a term. To add to your confusion, we *do* consider $\sigma\, \Gamma$ a set if $\Gamma = \{t_1, \ldots, t_n\}$ is a set of terms, defined in the obvious way by $\sigma\, \Gamma = \{\sigma\, t_1, \ldots, \sigma\, t_n\}$.

Now we are finally in a position to deal with the two most important notions of our kernel, *theorems* and *proofs*. A theorem is always considered with respect to a kernel state S, consisting of a set Γ of hypotheses together with a conclusion c, all of which are well-formed propositions with respect to S; and it is always accompanied by a proof P that tells us which rules were used in what order to wrest the theorem from the kernel. We write such a theorem/proof combination

$$P \rightsquigarrow_S \Gamma \vdash c \quad \text{or, if } \Gamma \text{ is the empty set} \quad P \rightsquigarrow_S c.$$

So the above notation implies:

- $\Gamma \subseteq \texttt{prop}_S$, $c \in \texttt{prop}_S$; also all the term and types mentioned in P are well-formed relative to S;
- S is the kernel state that is obtained as a side-effect of *replaying* the proof P starting in the *initial state* of the HOL-kernel;
- the theorem $\Gamma \vdash c$ can be deduced using the rules of the kernel, which again can be checked by replaying the proof P.

TODO:

- Explain better: The initial state of the HOL-kernel is not clearly defined; it can vary from system to system, and therefore we also do not fix it.

– define tfrees (t)
– define TyDef, TyIntro, side-effects of them and of Def
– side-effects that do not

$$\frac{u \in \mathtt{term}_S}{\textsc{Refl}\,(u) \rightsquigarrow_S u \doteq u} \qquad \frac{a \in \mathtt{prop}_S}{\textsc{Assume}\,(a) \rightsquigarrow_S \{a\} \vdash a}$$

$$\frac{P \rightsquigarrow_S \Gamma \vdash u \doteq (\lambda\, x.\, b) \cdot t}{\textsc{Beta}\,(P) \rightsquigarrow_S \Gamma \vdash u \doteq \langle\, x \mapsto t \,\rangle\, b} \qquad \frac{P \rightsquigarrow_S \Gamma \vdash u \doteq \lambda\, x.\, f \cdot x \qquad x \notin \mathrm{frees}\,(f)}{\textsc{Eta}\,(P) \rightsquigarrow_S \Gamma \vdash u \doteq f}$$

$$\frac{P \rightsquigarrow_S \Gamma \vdash a \doteq b \qquad Q \rightsquigarrow_S \Gamma' \vdash b \doteq c}{\textsc{Trans}\,(P, Q) \rightsquigarrow_S \Gamma \cup \Gamma' \vdash a \doteq c} \qquad \frac{P \rightsquigarrow_S \Gamma \vdash a \doteq b}{\textsc{Sym}\,(P) \rightsquigarrow_S \Gamma \vdash b \doteq a}$$

$$\frac{P \rightsquigarrow_S \Gamma \vdash a \mathrel{\dot{\longrightarrow}} b \qquad Q \rightsquigarrow_S \Gamma' \vdash a}{\textsc{Mp}\,(P, Q) \rightsquigarrow_S \Gamma \cup \Gamma' \vdash b} \qquad \frac{P \rightsquigarrow_S \Gamma \vdash a \doteq b \qquad Q \rightsquigarrow_S \Gamma' \vdash a}{\textsc{EqMp}\,(P, Q) \rightsquigarrow_S \Gamma \cup \Gamma' \vdash b}$$

$$\frac{P \rightsquigarrow_S \Gamma \vdash a \doteq b}{\textsc{EqImp}\,(P) \rightsquigarrow_S \Gamma \vdash a \mathrel{\dot{\longrightarrow}} b} \qquad \frac{P \rightsquigarrow_S \Gamma \vdash a \mathrel{\dot{\longrightarrow}} b \qquad Q \rightsquigarrow_S \Gamma' \vdash b \mathrel{\dot{\longrightarrow}} a}{\textsc{ImpAntiSym}\,(P, Q) \rightsquigarrow_S \Gamma \cup \Gamma' \vdash b \doteq a}$$

$$\frac{P \rightsquigarrow_S \Gamma \vdash a \qquad Q \rightsquigarrow_S \Gamma' \vdash b}{\textsc{Conj}\,(P, Q) \rightsquigarrow_S \Gamma \cup \Gamma' \vdash a \mathbin{\dot{\wedge}} b} \qquad \frac{P \rightsquigarrow_S \Gamma \vdash a \mathbin{\dot{\wedge}} b}{\textsc{Conj1}\,(P) \rightsquigarrow_S \Gamma \vdash a} \qquad \frac{P \rightsquigarrow_S \Gamma \vdash a \mathbin{\dot{\wedge}} b}{\textsc{Conj2}\,(P) \rightsquigarrow_S \Gamma \vdash b}$$

$$\frac{P \rightsquigarrow_S \Gamma \vdash a \mathrel{\dot{\longrightarrow}} \dot{0}}{\textsc{NotI}\,(P) \rightsquigarrow_S \Gamma \vdash \dot{\neg} a} \qquad \frac{P \rightsquigarrow_S \Gamma \vdash \dot{\neg} a}{\textsc{NotE}\,(P) \rightsquigarrow_S \Gamma \vdash a \mathrel{\dot{\longrightarrow}} \dot{0}}$$

$$\frac{P \rightsquigarrow_S \Gamma \vdash a \qquad b \in \mathtt{prop}_S}{\textsc{Disj1}\,(P, b) \rightsquigarrow_S \Gamma \vdash a \mathbin{\dot{\vee}} b} \qquad \frac{P \rightsquigarrow_S \Gamma \vdash b \qquad a \in \mathtt{prop}_S}{\textsc{Disj2}\,(P, a) \rightsquigarrow_S \Gamma \vdash a \mathbin{\dot{\vee}} b}$$

$$\frac{P \rightsquigarrow_S \Gamma \vdash a \mathbin{\dot{\vee}} b \qquad Q \rightsquigarrow_S \Gamma' \vdash c \qquad R \rightsquigarrow_S \Gamma'' \vdash c}{\textsc{DisjCases}\,(P, Q, R) \rightsquigarrow_S \Gamma \cup (\Gamma' \mathbin{\tilde{-}} a) \cup (\Gamma'' \mathbin{\tilde{-}} b) \vdash c}$$

$$\frac{P \rightsquigarrow_S \Gamma \vdash c \qquad h \in \mathtt{prop}_S}{\textsc{Disch}\,(P, h) \rightsquigarrow_S \Gamma \mathbin{\tilde{-}} h \vdash h \mathrel{\dot{\longrightarrow}} c} \qquad \frac{P \rightsquigarrow_S \Gamma \vdash \dot{0} \qquad a \in \mathtt{prop}_S}{\textsc{Contr}\,(P, a) \rightsquigarrow_S \Gamma \mathbin{\tilde{-}} \dot{\neg} a \vdash a}$$

$$\frac{t \in \mathtt{prop}_S}{\textsc{Bool}\,(t) \rightsquigarrow_S (t \doteq \dot{0}) \mathbin{\dot{\vee}} (t \doteq \dot{1})} \qquad \frac{P \rightsquigarrow_S \Gamma \vdash \mathtt{inj}\, f \qquad Q \rightsquigarrow_S \Gamma' \vdash f \cdot x \doteq f \cdot y}{\textsc{InjE}\,(P, Q, x, y) \rightsquigarrow_S \Gamma \cup \Gamma' \vdash x \doteq y}$$

$$\frac{P \rightsquigarrow_S \Gamma \vdash (f \cdot x \doteq f \cdot y) \mathrel{\dot{\longrightarrow}} (x \doteq y) \qquad x, y \in \mathtt{var}_S \qquad x \neq y \qquad x, y \notin \mathrm{frees}\,(\Gamma, f)}{\textsc{InjI}\,(P, f) \rightsquigarrow_S \Gamma \vdash \mathtt{inj}\, f}$$

$$\frac{f \in \mathtt{term}_S \qquad x, y \in \mathtt{var}_S \qquad x \neq y \qquad x, y \notin \mathrm{frees}\,(f) \qquad \tau\,(f) = \tau\,(x) \to \tau(y)}{\textsc{Surj}\,(f) \rightsquigarrow_S \mathtt{surj}\, f \doteq \dot{\forall}\, y.\, \dot{\exists}\, x.\, f \cdot x \doteq y}$$

$$\frac{\Gamma \rightsquigarrow_S P \vdash p \cdot t}{\textsc{Select}\,(P, p, t) \rightsquigarrow_S \Gamma \vdash p \cdot (\dot{\varepsilon}\, p)}$$

Fig. 8. Rules without binders

$$\frac{P \rightsquigarrow_S \Gamma \vdash f \doteq g \qquad Q \rightsquigarrow_S \Gamma' \vdash u \doteq v}{\textsc{MkComb}\,(P, Q) \rightsquigarrow_S \Gamma \cup \Gamma' \vdash f \cdot u \doteq g \cdot v}$$

$$\frac{P \rightsquigarrow_S \Gamma \vdash u \doteq v \qquad x \in \mathtt{var}_S \qquad x \notin \mathrm{frees}\,(\Gamma)}{\textsc{Abs}\,(P, x) \rightsquigarrow_S \Gamma \vdash \lambda\, x.\, u \doteq \lambda\, x.\, v}$$

$$\frac{P \rightsquigarrow_S \Gamma \vdash \dot{\forall}\, x.\, p \qquad t \in \mathtt{term}_S \qquad \tau\,(t) = \tau\,(x)}{\textsc{Spec}\,(P, t) \rightsquigarrow_S \Gamma \vdash \langle\, x \mapsto t \,\rangle\ p}$$

$$\frac{P \rightsquigarrow_S \Gamma \vdash c \qquad x \in \mathtt{var}_S \qquad x \notin \mathrm{frees}\,(\Gamma)}{\textsc{Gen}\,(P, x) \rightsquigarrow_S \Gamma \vdash \dot{\forall}\, x.\, c}$$

$$\frac{\sigma = \langle\, x \mapsto t \,\rangle \in \mathtt{subst}_S \qquad a \in \mathtt{term}_S \qquad P \rightsquigarrow_S \Gamma \vdash \sigma\, a}{\textsc{Exists}\,(P, \dot{\exists}\, x.\, a,\, t) \rightsquigarrow_S \Gamma \vdash \dot{\exists}\, x.\, a}$$

$$\frac{P \rightsquigarrow_S \Gamma \vdash \dot{\exists}\, x.\, a \qquad Q \rightsquigarrow_S \Gamma' \vdash c \qquad v \in \mathtt{var}_S \qquad v \notin \mathrm{frees}\,(\Gamma' \tilde{-} \langle\, x \mapsto v \,\rangle\ a,\, c)}{\textsc{Choose}\,(v, P, Q) \rightsquigarrow_S \Gamma \cup (\Gamma' \tilde{-} \langle\, x \mapsto v \,\rangle\ a) \vdash c}$$

$$\frac{P \rightsquigarrow_S \Gamma \vdash u \doteq v \qquad x_1, \ldots, x_n \in \mathtt{var}_S \qquad \{x_1, \ldots, x_n\} \cap \mathrm{frees}\,(\Gamma) = \emptyset}{\textsc{GenAbs}\,(P, x_1, \ldots, x_n) \rightsquigarrow_S \Gamma \vdash \lambda\, x_1.\, \ldots \lambda\, x_n.\, u \doteq \lambda\, x_1.\, \ldots \lambda\, x_n.\, v}$$

$$\frac{\begin{array}{c} P \rightsquigarrow_S \Gamma \vdash u \doteq v \qquad x_1, \ldots, x_n \in \mathtt{var}_S \qquad \{x_1, \ldots, x_n\} \cap \mathrm{frees}\,(\Gamma) = \emptyset \qquad A \notin \mathtt{tvar} \\ q = \mathtt{Const}\ \big(c, (A \to B) \to B'\big) \in \mathtt{term}_S \qquad q\,(\lambda\, x_1.\, \ldots q\,(\lambda\, x_n.\, u)) \in \mathtt{term}_S \end{array}}{\textsc{GenAbs}\,(P, q, x_1, \ldots, x_n) \rightsquigarrow_S \Gamma \vdash q\,(\lambda\, x_1.\, \ldots q\,(\lambda\, x_n.\, u)) \doteq q\,(\lambda\, x_1.\, \ldots q\,(\lambda\, x_n.\, v))}$$

$$\frac{\begin{array}{c} P \rightsquigarrow_S \Gamma \vdash u \doteq v \qquad x_1, \ldots, x_n \in \mathtt{var}_S \qquad \{x_1, \ldots, x_n\} \cap \mathrm{frees}\,(\Gamma) = \emptyset \\ q = \mathtt{Const}\ (c, T) \in \mathtt{term}_S \qquad T = (\alpha \to B) \to B' \qquad \alpha \in \mathtt{tvar} \\ \underset{i=1,\ldots,n}{\forall}\, q_i = \mathtt{Const}\ (c, \langle\, \alpha \mapsto \tau\,(x_i) \,\rangle\ T) \in \mathtt{term}_S \qquad q_1\,(\lambda\, x_1.\, \ldots q_n\,(\lambda\, x_n.\, u)) \in \mathtt{term}_S \end{array}}{\textsc{GenAbs}\,(P, q, x_1, \ldots, x_n) \rightsquigarrow_S \Gamma \vdash q_1\,(\lambda\, x_1.\, \ldots q_n\,(\lambda\, x_n.\, u)) \doteq q_1\,(\lambda\, x_1.\, \ldots q_n\,(\lambda\, x_n.\, v))}$$

Fig. 9. Higher-order rules

$$\frac{P \rightsquigarrow_S \Gamma \vdash c \qquad \sigma = \langle\, \alpha_1 \mapsto T_1, \ldots,\, \alpha_n \mapsto T_n \,\rangle \in \mathtt{tsubst}_S}{\textsc{InstType}\,(P, \alpha_1, T_1, \ldots, \alpha_n, T_n) \rightsquigarrow_S \sigma\,\Gamma \vdash \sigma\, c}$$

$$\frac{P \rightsquigarrow_S \Gamma \vdash c \qquad \sigma = \langle\, x_1 \mapsto t_1, \ldots,\, x_n \mapsto t_n \,\rangle \in \mathtt{subst}_S}{\textsc{Inst}\,(P, x_1, t_1, \ldots, x_n, t_n) \rightsquigarrow_S \sigma\,\Gamma \vdash \sigma\, c}$$

$$\frac{\begin{array}{c} P_1 \rightsquigarrow_S \Gamma_1 \vdash l_1 \doteq r_1 \quad \ldots \quad P_n \rightsquigarrow_S \Gamma_n \vdash l_n \doteq r_n \qquad t \in \mathtt{term}_S \\ \underset{\gamma=l,r}{\forall}\, \sigma_\gamma = \langle\, x_1 \mapsto \gamma_1, \ldots,\, x_n \mapsto \gamma_n \,\rangle \in \mathtt{subst}_S \qquad P \rightsquigarrow_S \Gamma \vdash \sigma_l\, t \end{array}}{\textsc{Subst}\,(P_1, \ldots, P_n,\, \lambda\, x_1.\, \ldots \lambda\, x_n.\, t, P) \rightsquigarrow_S \Gamma \cup \Gamma_1 \cup \ldots \cup \Gamma_n \vdash \sigma_r\, t}$$

Fig. 10. Substitution rules

$$\frac{s \in \mathtt{string} \qquad c \in \mathtt{prop}_S}{\textsc{Axiom}\,(s, c) \rightsquigarrow_S c} \qquad \frac{s \in \mathtt{tag} \qquad h_1, \ldots, h_n, c \in \mathtt{prop}_S}{\textsc{Oracle}\,(s, h_1, \ldots, h_n, c) \rightsquigarrow_S \{h_1, \ldots, h_n\} \vdash c}$$

$$\frac{\begin{array}{c} s \in \mathtt{tag} \qquad h_1, \ldots, h_n \in \mathtt{prop}_S \qquad c \in \mathtt{prop}_S \\ t_1, \ldots, t_u \in \mathtt{term}_S \qquad T_1, \ldots, T_v \in \mathtt{type}_S \qquad P_1 \rightsquigarrow_S \Gamma_1 \vdash c_1 \quad \ldots \quad P_w \rightsquigarrow_S \Gamma_w \vdash c_w \end{array}}{\textsc{Method}\,(s, h_1, \ldots, h_n, c, t_1, \ldots, t_u, T_1, \ldots T_v, P_1, \ldots, P_w) \rightsquigarrow_S \{h_1, \ldots, h_n\} \vdash c}$$

Fig. 11. Unsafe rules

$$\frac{P \rightsquigarrow_S \{h_1, \ldots, h_n\} \vdash c \qquad c', h'_1, \ldots, h'_n \in \mathtt{term}_S \qquad c \cong c' \qquad \underset{i=1,\ldots,n}{\forall}\, h_i \cong h'_i}{P \rightsquigarrow_S \{h'_1, \ldots, h'_n\} \vdash c'}\ (\textsc{Modulo})$$

Fig. 12. Rule for proving modulo $\cong$

$$\frac{\begin{array}{c} c \in \mathtt{id} \qquad c \notin \mathrm{dom}\,(ctype_S) \qquad t \in \mathtt{term}_S \\ \mathrm{frees}\,(t) = \emptyset \qquad \mathrm{tfrees}\,(t) = \mathrm{tfrees}\,(\tau(t)) \end{array}}{\textsc{Def}\,(c, t) \rightsquigarrow_{S'} \mathtt{Const}\,(c, \tau(t)) \doteq t}\ (\textsc{Def})$$

Fig. 13. State-changing rules (TO DO: document TyDef, TyIntro)