GFOL: A Term-Generic Logic for Defining λ-Calculi *

Andrei Popescu
University of Illinois at Urbana-Champaign
popescu2@cs.uiuc.edu

Grigore Roşu
University of Illinois at Urbana-Champaign
grosu@cs.uiuc.edu

Abstract

Generic first-order logic (GFOL) is a first-order logic parameterized with terms defined axiomatically (rather than constructively), by requiring them to only provide generic notions of *free variable* and *substitution* satisfying reasonable properties. GFOL has a complete Gentzen system generalizing that of FOL. An important fragment of GFOL, called HORN^2, possesses a much simpler Gentzen system, similar to traditional context-based derivation systems of λ-calculi. HORN^2 appears to be sufficient for defining virtually any λ-calculi (including polymorphic and type-recursive ones) as *theories* inside the logic. GFOL endows its theories with a default loose semantics, complete for the specified calculi.

Categories and Subject Descriptors F.4.1 [*Mathematical Logic*]: Lambda calculus and related systems, Model theory, Proof theory

General Terms Theory, Languages

Keywords generic first-order logic, substitution, λ-calculus

1. Introduction

First-order logic (FOL) is one of the best-established logics in computer science. The models of FOL, called first-order structures, as well as its complete Gentzen deduction system are well understood and intuitive, thus making FOL an attractive formalism with many applications in specification and verification of systems, data-bases, automated reasoning, etc. Problems can be represented as *FOL theories*, i.e., as sets of FOL formulae over corresponding operational and relational symbols; then FOL provides, in a uniform way, appropriate *models* together with *complete deduction* rules.

FOL does not allow variables to be bound in terms (but only in formulae, via quantifiers), thus providing a straightforward notion of substitution in terms. On the other hand, most calculi that are used in the domain of programming languages, and not only, are crucially based on the notion of *binding* of variables *in terms*: terms "export" only a subset of their variables, their *free variables*, that can be substituted. Because of their complex formulation for terms, these calculi cannot be naturally defined as FOL theories. Consequently, they need to define their own models and deduction rules, and to state their own theorems of completeness, not always easy to prove. In other words, they are presented as entirely *new logics*, as opposed to *theories in an existing logic*, thus incurring all the drawbacks (and boredom) of repeating definitions and proofs following generic, well-understood patterns, but facing new "details".

In this paper we define *term-generic first-order logic*, or simply *generic first-order logic (GFOL)*, as a first-order logic parameterized by any terms that come with abstract notions of *free variable* and *substitution*. More precisely, in GFOL terms are elements in a generic set *Term* including a subset *Var* whose elements are called variables, that comes with functions $FV : Term \to \mathcal{P}_f(Var)$ and $Subst : Term \times Term^{Var} \to Term$ for *free variables* and *substitution*, respectively, that satisfy some expected properties. GFOL models provide interpretations of terms that satisfy, again, some reasonable properties. We show that GFOL admits a *complete* Gentzen-like deduction system, which is syntactically very similar to that of FOL; its proof of completeness modifies the classic proof of completeness for FOL to use the generic notions of term, free variables, substitutions and their generic properties. Extensions of GFOL with equality and with multiple sorts are also discussed.

By not committing to any particular definition of term, GFOL can be instanciated to different types of terms, such as, e.g., standard FOL terms, or λ-terms, or different categories of typed λ-terms, etc. When instanciated to standard FOL terms, GFOL becomes, as expected, precisely FOL. However, when instanciated to more complex terms, e.g., the terms of λ-calculus, GFOL becomes a logic where a particular calculus is a particular theory. For example, the GFOL formulae for extensionality in untyped λ-calculus and for typing of abstractions in simply-typed λ-calculus can be

$$\forall x, y.(\forall z.x\ z = y\ z) \Rightarrow x = y \quad \text{and}$$
$$(\forall x.typeOf(x,t) \Rightarrow typeOf(X,t')) \Rightarrow typeOf(\lambda x\!:\!t.X, t \to t')$$

where x, y, z and t, t' denote data and type variables, respectively, and X denotes a data term. This way, a specification of a calculus in GFOL brings a *meaningful complete semantics* for that calculus, because the axioms are stated *about* some models, the content of the axioms making the models "desirable". Indeed, GFOL models are "blank" models in the sense that they are only required to interpret the terms in a way that is consistent with substitution – it is the axioms that "personalize" the models; e.g., the above typing rule asks that, in any desirable model, $\lambda x\!:\!t.X$ has type $t \to t'$ whenever X has type t' independently of the choice of x of type t.

Even though the completeness (being equivalent to semi-decidability) of a fragment of a logic (whose syntax is decidable) follows from the completeness of the richer logic, there are good reasons to develop complete proof systems for certain particular sublogics as well. Besides a better understanding and self-containment of the sublogic, one important reason is the *granularity of proofs*. Indeed, proofs of goals in the sublogic that use the proof-system of the larger logic may be rather long and "junkish" and may look artificial in the context of the sublogic. For example, equational logic admits a very intuitive complete proof system [4], that simply "replaces equals by equals", thus avoiding the more intricate first-order proofs. An important goal of this paper is to

* Supported by NSF grants CCF-0234524, CCF-0448501, CNS-0509321.

also investigate conditions under which sublogics of GFOL admit specialized coarse-granularity proof systems.

It appears that a certain fragment of GFOL, that we call HORN^2, is sufficient for calculi-specification purposes. HORN^2 consists of GFOL sentences of the form

$$\forall \overline{y}.(\forall \overline{x}.\textstyle\bigwedge_{i=1}^{n} a_i(\overline{x},\overline{y}) \Rightarrow b_i(\overline{x},\overline{y})) \Rightarrow c(\overline{y})$$

with a_i, b_i, c atomic formulae ($\overline{x}$ and $\overline{y}$ denote tuples of variables), i.e., generalized Horn implications whose conditions are themselves Horn implications. We show that, under a certain reasonable proof-theoretic restriction that we call *amenability*, a HORN^2 theory admits a *complete* Gentzen system that "implements" each HORN^2 formula as above into a deduction rule of the form

$$\frac{\Gamma, a_i(\overline{z},\overline{T}) \rhd b_i(\overline{z},\overline{T}) \text{ for any } i = 1 \ldots n}{\Gamma \rhd c(\overline{T})}$$

where $\overline{z}$ is a fresh tuple of variables replacing $\overline{x}$, and $\overline{T}$ is a tuple of terms substituting $\overline{y}$.

Completeness of HORN^2 has, as particular cases, term-generic versions of completeness results for conditional-equational [4], Horn [17], and extensional [24] logics. Even more importantly, this completeness result qualifies HORN^2 as a *higher-level notation* for describing derivation systems for calculi, as it enables one to faithfully *recover* the original proof systems of the specified calculi in a uniform way, from their HORN^2 theories. For instance, the HORN^2 deduction rule corresponding to the previously mentioned typing axiom is the familiar context-based typing rule for abstractions:

$$\frac{\Gamma, \mathit{typeOf}(z,t) \rhd \mathit{typeOf}(T,t')}{\Gamma \rhd \mathit{typeOf}(\lambda z\!:\!t.T, t \to t')}, \text{ where } z \text{ is fresh w.r.t. } \Gamma.$$

(Above, we viewed the type declaration $z:t$ as an atomic formula $\mathit{typeOf}(z,t)$, of the same category with $\mathit{typeOf}(\lambda z\!:\!t.T, t \to t')$; note that the freshness assumption coincides with the usual requirement that z does not occur on the left of a type declaration in Γ, in order to avoid conflicts.) The HORN^2 notation is not only compact and syntactic-detail-free (like the one advocated by HOAS [19]), but also, by its very nature, *model-theoretically meaningful*.

Our main two contributions in this paper are:

- We show that the development of first-order logic is largely orthogonal to the particular syntax of terms. While previous work dealing with general terms models binding *explicitly*, we develop a logic, GFOL, that abstracts away bindings by considering terms as "black-boxes" that export substitutions and free variables; all particular known terms with bindings satisfy the GFOL generic axioms; and
- We provide a convenient notation and intuition for defining λ-calculi, that encourages a semantic specification style. GFOL endows the specified calculi with a *default complete semantics*, via the GFOL models of their defining theory.

Regarding the latter point, an early disclaimer is in order. The semantics that GFOL brings for the specified calculi falls into the category of *loose*, or *logical* semantics. Examples of loose semantics for λ-calculi include: (so called) "syntactic" models for untyped λ-calculus, Henkin models for simply-typed λ-calculus, Kripke-style models for recursive types, and Girard's qualitative domains and Bruce-Meyer-Mitchell models for System F, not to mention all their categorical variants. For extensive presentations of these and many other loose semantics, we recommend the monographs [3, 10, 15]. For a particular calculus defined as a GFOL theory, the implicit GFOL semantics has all the advantages, but, naturally, also all the drawbacks, of loose semantics. It is not the concern of this paper to advocate for a loose or for a fixed-model semantics, especially because we believe that there is no absolute answer. What we consider to be a particularly appealing aspect of GFOL semantics though is its uniform, *calculus-independent* nature. And the "general-purpose" GFOL semantics tends to be equivalent to the "domain-specific" ones developed for various calculi.

The remainder of this paper is structured as follows. Section 2 introduces GFOL (syntax, models and some properties), its fragment HORN^2, and complete Gentzen systems for them. Section 3 presents and discusses specifications of various λ-calculi. Section 4 compares, for untyped λ-calculus and System F, their ad hoc complete semantics already defined in the literature with this GFOL semantics obtained "automatically" from their definition as HORN^2 theories. Section 5 discusses related work, making room for GFOL in the extensively studied field of general approaches to representing λ-calculi. A short concluding section ends the paper.

We have exiled all the proofs into Appendix E.

2. Term-Generic First-Order Logic

We introduce a generic notion of first-order term, axiomatized by means of free variables and substitution, purposely not committing to any concrete syntax for terms. Then we show our first novel result in this paper, namely that the development of first-order logic essentially does not depend on the syntax of terms, but *only on the properties of substitution*. Additionally, as shown in Section 3, various forms of typed and untyped λ-calculi naturally fall into our framework by properly instanciating the generic notion of term (and implicitly of free variable and substitution). To keep the discussion notationally simple, we first develop the logic in an unsorted form and without equality, an later sketch an order-sorted extension.

2.1 Term Syntax

DEFINITION 1. *Let Var be a countably infinite set of variables. A* term syntax *over Var consists of the following data:*

(a) A countably infinite set Term such that Var $\subseteq$ *Term, whose elements are called* terms*;*

(b) A mapping FV : *Term* $\to \mathcal{P}_f(Var)$*; elements of FV*(T) *are called* free variables*, or simply* variables*, of* T*;*

(c) A mapping Subst : *Term* $\times$ *Term*$^{Var} \to$ *Term.*

These are subject to the following requirements ($x, T, T', \theta, \theta'$ *denote arbitrary variables, terms, and maps in Term*^{Var}*, respectively):*

(1) $\mathit{Subst}(x,\theta) = \theta(x)$*;*

(2) $\mathit{Subst}(T, 1_{Var}) = T$*;* [1]

(3) If $\theta|_{FV(T)} = \theta'|_{FV(T)}$*, then* $\mathit{Subst}(T,\theta) = \mathit{Subst}(T,\theta')$*;*

(4) $\mathit{Subst}(\mathit{Subst}(T,\theta),\theta') = \mathit{Subst}(T,\theta;\theta')$*, where for each* $x \in Var$*,* $(\theta;\theta')(x)$ *is, by definition,* $\mathit{Subst}(\theta(x),\theta')$*;*

(5) $FV(x) = \{x\}$*;*

(6) $FV(\mathit{Subst}(T,\theta)) = \bigcup\{FV(\theta(x)) : x \in FV(T)\}$.

From here on we may write a term syntax as a tuple (*Term*, *Var*, *FV*, *Subst*) or even just *Term* if the other components of the tuple are understood from context. Note that we assume the notion of term coming together with a notion of substitution which is *composable* (condition (4) above). Therefore, in our examples of calculi with bindings, we shall consider α*-equivalence classes* of terms rather than bare terms, a reasonable assumption when working at the logical, and not the implementation, level. For *distinct* variables $x_1, \ldots, x_n$, we write $[T_1/x_1, \ldots, T_n/x_n]$ for the function *Var* $\to$ *Term* that maps each x_i to T_i for $i = \overline{1,n}$ and all the other variables to themselves, and $T[T_1/x_1, \ldots, T_n/x_n]$ for $\mathit{Subst}(T, [T_1/x_1, \ldots, T_n/x_n])$.

PROPOSITION 1. *The following hold:*

1. $x \notin FV(T)$ *implies* $T[T'/x] = T$*;*

[1] Here and elsewhere, by language abuse, we let 1_{Var} denote the inclusion mapping of *Var* into *Term*.

2. $y[T/x] = T$ *if* $y = x$ *and* $y[T/x] = y$ *otherwise;*
3. $FV(T[T'/x]) = FV(T) \setminus \{x\} \cup FV(T')$*;*
4. $T[y/x][z/y] = T[z/x]$ *if* $y \notin FV(T)$*;*
5. $T[y/x][x/y] = T$ *if* $y \notin FV(T)$*.*

2.2 First-Order Logic over a Term Syntax

DEFINITION 2. *A* generic first-order language *consists of a term syntax* $(Term, Var, FV, Subst)$ *together with a countable ranked set* $\Pi = (\Pi_n)_{n \in I\!N}$*, of* relation symbols.

If *Term* is a term syntax as above, then we write generic first-order languages as $(Term, \Pi)$; if more components of the term syntax are relevant for a given context, then we can also mention them in the tuple, e.g., $(Term, Var, \Pi)$ or $(Term, Var, FV, \Pi, Subst)$, etc.

DEFINITION 3. *A GFOL* model *is a triple* $(A, (A_T)_{T \in Term}, (A_\pi)_{\pi \in \Pi})$*, such that:*

(a) A *is a set, called the* carrier set.
(b) For each $T \in Term$*,* A_T *is a mapping* $A^{Var} \to A$ *such that:*
  (i) $A_x(\rho) = \rho(x)$*;*
  (ii) For each ρ *and* ρ' *such that* $\rho \restriction_{FV(T)} = \rho' \restriction_{FV(T)}$*, it holds that* $A_T(\rho) = A_T(\rho')$*;*
  (iii) $A_{Subst(T,\theta)}(\rho) = A_T(A_\theta(\rho))$*, where for each* $\theta \in Term^{Var}$*,* $A_\theta : A^{Var} \to A^{Var}$ *is defined by* $A_\theta(\rho)(x) = A_{\theta(x)}(\rho)$*.*
(c) For each $\pi \in \Pi_n$*,* A_π *is an* n*-ary relation on* A*.*

Note that, unlike in classical FOL models where the interpretation of terms is *built* from operations, in GFOL models the interpretation of terms is *assumed* (in the style of Henkin models). However, due to the axioms ruling these interpretations, when instanciated to FOL terms, GFOL yields essentially the same models.

Term can be organized as a model $(Term, (Term_T)_{T \in Term}, (Term_\pi)_{\pi \in \Pi})$ in many ways, corresponding to the choice of relations $Term_\pi$, by letting $Term_T(\rho)$ be $Subst(T, \rho)$. These are indeed models, since conditions (i)-(iii) from the model definition coincide in this case with (1), (3) and (4) in the term syntax definition. Any such model will be henceforth called a *(GFOL) Herbrand model*. If one defines *model homomorphisms* as expected, then one gets that the Herbrand model with all relations empty is *free* over X in the category of models and model homomorphisms. However, we shall not be interested in such categorical/algebraic aspects here.

Above, and from now on, we let x, x_i, y, u, v, etc., range over variables, T, T_i, T', etc., over terms, ρ, ρ', etc., over valuations in A^{Var}, and π, π', etc., over relation symbols. *Formulae* are defined the usual way, starting from *atomic formulae* $\pi(T_1, \ldots, T_n)$ and applying connectives $\wedge, \neg$ and quantifier $\forall$. We let *Formula* denote the set of formulae. For each formula φ, the set $A_\varphi \subseteq A^{Var}$, of *valuations that make* φ *true in* A, is defined recursively on the structure of formulae as follows:

- $\rho \in A_{\pi(T_1,\ldots,T_n)}$ iff $(A_{T_1}(\rho), \ldots, A_{T_n}(\rho)) \in A_\pi$;
- $\rho \in A_{\neg\varphi}$ iff $\rho \notin A_\varphi$;
- $\rho \in A_{\varphi \wedge \psi}$ iff $\rho \in A_\varphi$ and $\rho \in A_\psi$;
- $\rho \in A_{\forall x.\varphi}$ iff $\rho[x \leftarrow a] \in A_\varphi$ for all $a \in A$.

If $\rho \in A_\varphi$ we say that A *satisfies* φ *under valuation* ρ and write $A \models_\rho \varphi$. If $A_\varphi = A^{Var}$ we say that A *satisfies* φ and write $A \models \varphi$. Given a set of formulae Γ, $A \models \Gamma$ iff $A \models \varphi$ for all $\varphi \in \Gamma$. Above, and from now on, we let φ, ψ, χ range over arbitrary formulae and A over arbitrary models.

DEFINITION 4. *For each formula* φ*, the set* $FV(\varphi)$*, of its* free variables*, is defined recursively as follows:*

- $FV(\pi(T_1, \ldots, T_n)) = FV(T_1) \cup \ldots \cup FV(T_n)$*;*
- $FV(\neg\varphi) = FV(\varphi)$*;*
- $FV(\varphi \wedge \psi) = FV(\varphi) \cup FV(\psi)$*;*
- $FV(\forall x.\varphi) = FV(\varphi) \setminus \{x\}$*.*

Note that, since $FV(T)$ is finite for each term T, $FV(\varphi)$ is also finite for each formula φ. A *sentence* is a closed formula φ, i.e., one with $FV(\varphi) = \emptyset$. A *theory*, or a *specification*, is a set of sentences.

DEFINITION 5. Substitution of terms for variables in formulae, $Subst : Formula \times Term^{Var} \to Formula$*, is defined as follows:*

- $Subst(\pi(T_1, \ldots, T_n), \theta) = \pi(Subst(T_1, \theta), \ldots, Subst(T_n, \theta))$*;*
- $Subst(\neg\varphi, \theta) = \neg Subst(\varphi, \theta)$*;*
- $Subst(\varphi \wedge \psi, \theta) = Subst(\varphi, \theta) \wedge Subst(\psi, \theta)$*;*
- $Subst(\forall x.\varphi, \theta) = \forall z.Subst(\varphi, \theta[x \leftarrow z])$*, where* z *is the least variable*[2] *not in* $FV(\varphi) \cup \bigcup\{\theta(y) : y \in FV(\varphi)\}$*.*

For substitution in formulae we adopt notational conventions similar to the ones about substitution in terms, e.g., $\varphi[T/x]$.

DEFINITION 6. α-equivalence of formulae*, written* $\equiv_\alpha$*, is defined to be the least relation* $R \subseteq Formula \times Formula$ *satisfying:*

- $\pi(T_1, \ldots, T_n)\ R\ \pi(T_1, \ldots, T_n)$*;*
- $\neg\varphi\ R\ \neg\psi$ *if* $\varphi\ R\ \psi$*;*
- $\varphi \wedge \varphi'\ R\ \psi \wedge \psi'$ *if* $\varphi\ R\ \psi$ *and* $\varphi'\ R\ \psi'$*;*
- $\forall x.\varphi\, R\, \forall y.\psi$ *if* $\varphi[z/x]\, R\, \psi[z/y]$ *for some* $z \notin FV(\varphi) \cup FV(\psi)$*.*

Thus GFOL is a logic generic only w.r.t. terms - formulae are "concrete" first-order formulae over generic terms, with a "concrete" (and not generic) notion of α-equivalence, standardly constructed on top of the *identity* of terms and using the quantifier bindings; however, in concrete cases involving bindings, the identity of terms is itself α-equivalence w.r.t. term bindings.

For the following proposition, recall the definitions, for a model A and two mappings $\theta, \theta' : Var \to Term$, of the composition $\theta; \theta' : Var \to Term$ and of the mapping $A_\theta : A^{Var} \to A^{Var}$:

- $(\theta; \theta')(x) = Subst(\theta(x), \theta')$;
- $A_\theta(\rho)(x) = A_{\theta(x)}(\rho)$.

PROPOSITION 2. *The following hold:*

1. *If* $\rho \restriction_{FV(\varphi)} = \rho' \restriction_{FV(\varphi)}$*, then* $\rho \in A_\varphi$ *iff* $\rho' \in A_\varphi$*;*
2. $\rho \in A_{Subst(\varphi,\theta)}$ *iff* $A_\theta(\rho) \in A_\varphi$*;*
3. $\varphi \equiv_\alpha \psi$ *implies* $A_\varphi = A_\psi$*;*
4. $\varphi \equiv_\alpha \psi$ *implies* $FV(\varphi) = FV(\psi)$*;*
5. $\equiv_\alpha$ *is an equivalence;*
6. $\varphi \equiv_\alpha Subst(\varphi, 1_{Var})$*;*
7. $y \notin FV(\varphi)$ *implies* $\varphi[y/x][z/y] \equiv_\alpha \varphi[z/x]$*;*
8. $x \notin FV(\varphi)$ *implies* $\varphi[T/x] \equiv_\alpha \varphi$*;*
9. $\varphi \equiv_\alpha \psi$ *implies* $Subst(\varphi, \theta) \equiv_\alpha Subst(\psi, \theta)$*;*
10. $\theta \restriction_{FV(\varphi)} = \theta' \restriction_{FV(\varphi)}$ *implies* $Subst(\varphi, \theta) \equiv_\alpha Subst(\varphi, \theta')$*;*
11. $Subst(\varphi, \theta; \theta') \equiv_\alpha Subst(Subst(\varphi, \theta), \theta')$*;*
12. $\varphi \equiv_\alpha \varphi'$ *and* $\psi \equiv_\alpha \psi'$ *implies:* $\neg\varphi \equiv_\alpha \neg\varphi'$*,* $\varphi \wedge \psi \equiv_\alpha \varphi' \wedge \psi'$*,* $\forall x.\varphi \equiv_\alpha \forall x.\varphi'$*.*

Thus $\equiv_\alpha$ is an equivalence, preserves satisfaction and the free variables, and is compatible with substitution and language constructs (points (5), (3), (4), (9), (12) above). Hereafter, we shall identify formulae modulo α-equivalence, since mappings FV, $Subst$, $A\_$ and those that build formulae are well defined on equivalence classes.

[2] One may interpret "the least" as "having the least index", where we assume an indexing on the (countable) set of variables; we pick the least variable in order to make a choice - any variable with the mentioned property would do.

Examples

FOL. As expected, classical FOL is an instance of GFOL. Indeed, let (Var, Σ, Π) be a first-order language (possibly with equality - see Subsection 2.3), where *Var* is a countably infinite set of variables, and $\Sigma = (\Sigma_n)_{n\in\mathbb{N}}$ and $\Pi = (\Pi_n)_{n\in\mathbb{N}}$ are ranked sets of operation and relation symbols. Let *Term* be the term syntax consisting of ordinary first-order terms over Σ and *Var* with $FV : Term \to \mathcal{P}_f(Var)$ giving *all* the variables in each term (all variables are free in FOL terms) and $Subst : Term \times Term^{Var} \to Term$ the normal substitution on FOL terms (no precautions need to be taken here, since there is no variable capture). Define a generic first-order language as $(Term, \Pi)$. A classical FOL model $(A, (A_\sigma)_{\sigma\in\Sigma}, (A_\pi)_{\pi\in\Pi})$ yields a GFOL model $(A, (A_T)_{T\in Term}, (A_\pi)_{\pi\in\Pi})$ by defining the meaning of terms as derived operations. Conversely, from a GFOL model $(A, (A_T)_{T\in Term}, (A_\pi)_{\pi\in\Pi})$, one can extract a FOL model by defining $A_\sigma : A^n \to A$ as $A_\sigma(a_1, \ldots, a_n) = A_{\sigma(x_1,\ldots,x_n)}(\rho)$, where $x_1, \ldots, x_n$ are distinct variables and ρ is a valuation that maps each x_i to a_i. (The definition of A_σ does not depend on the choice of the x_i's.) The two model mappings are mutually inverse and preserve satisfaction. Thus, for this particular choice of terms, GFOL yields FOL.

A Formula-Typed Logic. FOL is a trivial instance of GFOL. However, GFOL terms may be arbitrarily exotic. Besides terms of various λ-calculi (that will be discussed in Section 3), one may also have terms that interfere with formulae in non-trivial ways, as shown by the following example, where terms may abstract variables having formulae as types. Let (Var, Σ, Π) be a first-order language and *Term* and *Formula* be defined mutually recursively:

$$\begin{array}{lll} Term & ::= & Var \mid \Sigma(Term, \ldots, Term) \mid Term\langle Term\rangle \\ & & \lambda\, Var : Formula.\, Term \\ Formula & ::= & Term = Term \mid \Pi(Term, \ldots, Term) \mid \neg Formula \\ & & Formula \wedge Formula \mid \forall\, Var.\, Formula \end{array}$$

where productions $Term ::= \Sigma(Term, \ldots, Term)$ and $Formula ::= \Pi(Formula, \ldots, Formula)$ have the restriction that the number of *Term*'s equals the rank of the corresponding operation in Σ or relation in Π. The free variables of terms are defined recursively by $FV(\lambda x : \varphi.T) = (FV(T) \cup FV(\varphi)) \setminus \{x\}$ and on the other term constructs as expected, and term α-equivalence and substitution as expected. It is easy to check that terms up to α-equivalence form a GFOL term syntax. Moreover, even though formulae were defined recursively *together* with the terms, they are still nothing but first-order formulae over the terms, hence they fall into the framework of GFOL. This logic can be seen as an "extremely-typed" λ-calculus, and is itself powerful enough to capture several forms of typed calculi.

2.3 GFOL with Equality

A *generic first-order language with equality* is a generic first-order language that has an emphasized binary relation symbol "=", interpreted in all models as the equality relation. All the other concepts remain the same.

GFOL with equality is an important variant of GFOL and will prove appropriate for conveniently capturing various λ-calculi, where equality plays a central role together with typing/kinding. Since typing will be defined as a binary relation which is implicitly compatible with equality in our framework, type preservation will hold by default in calculi specified in GFOL with equality (see Section 3.)

2.4 Many-Sorted and Order-Sorted GFOL

The notions of a term syntax and term-generic first-order languages have rather straightforward many-sorted and order-sorted generalizations. We next sketch an order-sorted version of GFOL, which also covers the many-sorted case. Order-sorted GFOL generalizes *order-sorted equational logic* [9, 22].

Let $S = (S, <)$ be a fixed poset. Elements of S are called *sorts*, and $<$ is called the *subsort* relation. We assume that any two sorts s, s' having a common subsort (i.e., a sort s'' with $s'' < s$ and $s'' < s'$), also have a *greatest common subsort*, denoted $s \wedge s'$. An *S-sorted set* is a family of sets $A = (A_s)_{s\in S}$ such that $A_s \subseteq A_{s'}$ whenever $s < s'$. Let $All(A)$ denote the set $\bigcup_{s\in S} A_s$. We call A *unambiguous* if $A_s \cap A_{s'} = A_{s\wedge s'}$ if s and s' have a common subsort, and $A_s \cap A_{s'} = \emptyset$ otherwise. Note that an unambiguous S-sorted set A can be recovered from $All(A)$ and the relation "has sort" between elements of $All(A)$ and S. Let $A = (A_s)_{s\in S}$ and $B = (B_s)_{s\in S}$ be two S-sorted sets. A is *included* in B, written $A \subseteq B$, if $A_s \subseteq B_s$ for all $s \in S$. An *S-sorted mapping* between A and B is a family of mappings $(h_s : A_s \to B_s)_{s\in S}$ such that h_s is a restriction and corestriction of $h_{s'}$ whenever $s < s'$. Let $Map(A, B)$ denote the set of S-sorted mappings between A and B. For convenience, we define the intersection of an S-sorted set $A = (A_s)_{s\in S}$ and an (unsorted) set D as $A \cap D = (A_s \cap D)_{s\in S}$.

Let *Var* be an unambiguous, sortwise countably infinite S-sorted set of variables. An *S-sorted term syntax* over *Var* consists of the following data:

(a) An unambiguous, sortwise countably infinite S-sorted set *Term* such that $Var \subseteq Term$;

(b) A mapping $FV : All(Term) \to \mathcal{P}_f(All(Var))$;

(c) A mapping $Subst : All(Term) \times All(Term)^{All(Var)} \to All(Term)$ such that, for each $s \in S$, $T \in Term_s$ and $\theta \in All(Term)^{All(Var)}$, $Subst(T, \theta) \in Term_s$,

subject to the following requirements (where $x, T, T', \theta, \theta'$ denote arbitrary variables, terms, and maps in $All(Term)^{All(Var)}$, respectively):

(1) $Subst(x, \theta) = \theta(x)$;

(2) $Subst(T, 1_{All(Var)}) = T$;

(3) If $\theta\restriction_{FV(T)} = \theta'\restriction_{FV(T)}$, then $Subst(T, \theta) = Subst(T', \theta)$;

(4) $Subst(Subst(T, \theta), \theta') = Subst(T, \theta; \theta')$, where for each $x \in Var$, $(\theta; \theta')(x)$ is, by definition, $Subst(\theta(x), \theta')$;

(5) $FV(x) = \{x\}$;

(6) $FV(Subst(T, \theta)) = \bigcup\{FV(\theta(x)) : x \in FV(T)\}$.

An *S-sorted generic first-order language* consists of the following: an unambiguous, sortwise countably infinite S-sorted set *Var*; an S-sorted term syntax *Term* over *Var*; a countable S^*-ranked set $\Pi = (\Pi_w)_{w\in S^*}$, of *relation symbols*. A GFOL *model* is a triple $(A, (A_T)_{T\in Term}, (A_\pi)_{\pi\in\Pi})$, such that:

(a) A is an S-sorted set;

(b) For each $\pi \in \Pi_{s_1\ldots s_n}$, $A_\pi \subseteq A_{s_1} \times \ldots \times A_{s_n}$;

(c) For each $T \in All(Term)$, A_T is a mapping $Map(Var, A) \to All(A)$ such that whenever $T \in Term_s$, $A_T(\rho) \in A_s$ for all $\rho \in Map(Var, A)$ and:

(i) If $x \in Var_s$, then $A_x(\rho) = \rho_s(x)$;

(ii) $A_{Subst(T,\theta)}(\rho) = A_T(A_\theta(\rho))$, where for each $\theta \in All(Term)^{All(Var)}$, $A_\theta : Map(Var, A) \to Map(Var, A)$ is defined by $A_\theta(\rho)_s(x) = A_{\theta(x)}(\rho)$ if $x \in Var_s$;

(iii) For each $\rho, \rho' \in Map(Var, A)$ such that $\rho\restriction_{Map(Var,A)\cap FV(T)} = \rho'\restriction_{Map(Var,A)\cap FV(T)}$, it holds that $A_T(\rho) = A_T(\rho')$.

Now first-order formulae are defined as usual. All the concepts and results about GFOL in this paper, including completeness of

various proof systems for various fragments of the logic, can be easily (but admittedly tediously) extended to the many-sorted and order-sorted cases.

2.5 GFOL Gentzen System and Completeness

Next we show that the axiomatic properties of the generic notions of free variable and substitution in GFOL provide enough infrastructure for proving generic versions of classical FOL results. We are interested in a completeness theorem here, but other model-theoretic results could be generalized as well. We shall use the same cut-free Gentzen system as the one usually given in the classical setting [7]. It is worth mentioning that our system rather *looks the same* than *is the same* to the classical one, since in the tables below T and $\_[T/x]$ denote *generic* terms and substitution.

We fix a generic first-order language (Term, Π). A *sequent* is a pair written $\Gamma \triangleright \Delta$, with Γ and Δ (at most) countable sets of formulae, called the *antecedent* and the *succedent* of the sequent. The sequent $\Gamma \triangleright \Delta$ is called *tautological* if for each model A, $\bigcap_{\varphi\in\Gamma} A_\varphi \subseteq \bigcup_{\psi\in\Delta} A_\psi$, and *falsifiable* if it is not tautological. A rule is a pair $\frac{H}{S}$ consisting of a sequent S and a (possibly empty) *list* of sequents H. If $H = \cdot$ (i.e., it is empty) we call $\frac{H}{S}$ an *axiom*. The notion of a *proof tree* for a Gentzen system is defined the usual way - its nodes are labelled with sequents, in a way that is consistent with the rules: if a node is labelled with S, then its descendants, if they exist, are labelled with the elements of H, where $\frac{H}{S}$ is a rule in the Gentzen system. A *completed* proof tree is one which has all its leaves labeled with axioms. A rule $\frac{H}{S}$ is *sound* if whenever all sequents in H are tautological, S is tautological too. A sequent is *provable* in a Gentzen system if it is the root of a completed proof tree. A Gentzen system is *sound*, if all its provable sequents are tautological, and *complete* if all tautological sequents are provable. Note that soundness of a Gentzen system is equivalent to soundness of each of its rules.

We consider the Gentzen system, say $\mathcal{G}$, given by the following rule schemes (we write $\Gamma\varphi$ instead of $\Gamma \cup \{\varphi\}$):

Left	Right	
$\frac{\Gamma \triangleright \Delta\ \varphi}{\Gamma \neg\varphi \triangleright \Delta}$	$\frac{\Gamma\ \varphi \triangleright \Delta}{\Gamma \triangleright \Delta\ \neg\varphi}$	$(\neg)$
$\frac{\Gamma\ \varphi\ \psi \triangleright \Delta}{\Gamma\ \varphi \wedge \psi \triangleright \Delta}$	$\frac{\Gamma \triangleright \Delta\ \varphi \quad \Gamma \triangleright \Delta\ \psi}{\Gamma \triangleright \Delta\ \varphi \wedge \psi}$	$(\wedge)$
$\frac{\Gamma\ \forall x.\varphi\ \varphi[T/x] \triangleright \Delta}{\Gamma\ \forall x.\varphi \triangleright \Delta}$	$\frac{\Gamma \triangleright \Delta\ \varphi[y/x] \quad y \text{ not free in } \Gamma, \Delta, \forall x.\varphi}{\Gamma \triangleright \Delta\ \forall x.\varphi}$	$(\forall)$
$\frac{\cdot}{\Gamma \triangleright \Delta}$ if $\Gamma \cap \Delta \neq \emptyset$		(Ax)

Given a theory E, a sequent $\Gamma \triangleright \Delta$ is called *E-tautological* if for each model $A \models E$, $\bigcap_{\varphi\in\Gamma} A_\varphi \subseteq \bigcup_{\psi\in\Delta} A_\psi$. The above Gentzen system is meant to entail *tautological*, i.e., $\emptyset$-tautological, GFOL sequents, and thus is not parameterized by any fixed theory E. Note however that, for a countable theory E, $\Gamma \triangleright \Delta$ is E-tautological iff $(\Gamma \cup E) \triangleright \Delta$ is tautological, and thus the case of a fixed countable theory E can be covered by adding E to the antecedent of the desired sequent. Gentzen systems specialized for fixed theories, that apply the axioms of the theory *directly* as rules, are discussed in Subsection 2.6.

THEOREM 1. *Gentzen system $\mathcal{G}$ is sound and complete for GFOL.*

To obtain a Gentzen system for GFOL with equality, we add to $\mathcal{G}$:

$$\frac{\Gamma \triangleright \Delta\ T = T}{\Gamma \triangleright \Delta} \quad \text{(Inst-Refl)}$$

$$\frac{\Gamma \triangleright \Delta\ T_1 = T_2 \ ,\ \Gamma\ T_2 = T_1 \triangleright \Delta}{\Gamma \triangleright \Delta} \quad \text{(Inst-Symm)}$$

$$\frac{\Gamma \triangleright \Delta\ T_1 = T_2 \ ,\ \Gamma \triangleright \Delta\ T_2 = T_3 \ ,\ \Gamma\ T_1 = T_3 \triangleright \Delta}{\Gamma \triangleright \Delta} \quad \text{(Inst-Trans)}$$

$$\frac{\Gamma \triangleright \Delta\ T_1 = T_1' \ ,\ \ldots \ ,\ \Gamma \triangleright \Delta\ T_n = T_n' \ ,\ \Gamma \triangleright \Delta\ \pi(T_1,\ldots,T_n) \ ,\ \Gamma\ \pi(T_1',\ldots,T_n') \triangleright \Delta}{\Gamma \triangleright \Delta} \quad \text{(Inst-Comp}_\pi\text{)}$$

$$\frac{\Gamma \triangleright \Delta\ T_1 = T_2 \ ,\ \Gamma\ T[T_1/x] = T[T_2/x] \triangleright \Delta}{\Gamma \triangleright \Delta} \quad \text{(Inst-Subst)}$$

We call these the *equality rules*. Note that the rules in the above two tables make full sense in our generic framework, since concrete syntax of terms is *not* required; all that is needed here are abstract notions of term and substitution. In concrete cases, congruence w.r.t. various kinds of operations will be captured as particular cases of (Inst-Subst). Let $\mathcal{G}_=$ be this enriched Gentzen system.

THEOREM 2. *$\mathcal{G}_=$ is sound and complete for GFOL with equality.*

The notation used for the equality rules, "(Inst-...)", comes from regarding these rules as *instances* of some axiom-schemes, namely:

$x = x$	(Refl)
$x = y \Rightarrow y = x$	(Symm)
$x = y \wedge y = z \Rightarrow x = z$	(Trans)
$(x_1 = y_1 \wedge \ldots \wedge x_n = y_n \wedge \pi(x_1,\ldots,x_n)) \Rightarrow \pi(y_1,\ldots,y_n)$	(Comp$_\pi$)
$x = y \Rightarrow T[z \leftarrow x] = T[z \leftarrow y]$	(Subst)

Let *Eql* be the set of these *equality axioms*. The relationship between "axioms" and their associated "rules" will be exploited in the following subsection, where we develop a more effective theory-oriented Gentzen system for a fragment of GFOL.

2.6 The HORN^2 Fragment of GFOL

We next consider a fragment of GFOL, called HORN^2 because it only allows formulae which are universally quantified implications whose conditions are themselves universally quantified implications of atomic formulae. All our GFOL specifications of calculi with bindings in Section 3.1 will consist of HORN^2 formulae. As shown in the sequel, we can associate to these theories more natural and intuitive proof systems, which resemble almost identically (modulo syntactic sugar modifications and some built-in type preservation properties) the corresponding original proof systems of the calculi.

For convenience, we assume that the language also contains the logical connectives $\top$ (zero-ary, corresponding to "true") and $\Rightarrow$ (binary, the logical implication) and that the Gentzen system $\mathcal{G}$ also contains the rules

$$\frac{\Gamma \triangleright \Delta\ \varphi \ ,\ \Gamma\ \psi \triangleright \Delta}{\Gamma\ \varphi \Rightarrow \psi \triangleright \Delta}, \qquad \frac{\Gamma\varphi \triangleright \Delta\ \psi}{\Gamma \triangleright \Delta\ \varphi \Rightarrow \psi}, \quad \text{and} \quad \frac{\cdot}{\Gamma \triangleright \top}.$$

$\top$ and $\Rightarrow$ can be treated as derived connectives and the above rules can be derived for them, but we prefer to take them here as primitives. We take the convention that $\top$ is an atomic formula. Clearly, $\mathcal{G}$ is still sound and complete.

In what follows, $\overline{x}$ denotes a tuple $(x_1,\ldots,x_n)$ of variables, $\overline{T}$ a tuple $(T_1,\ldots,T_n)$ of terms, and, for a formula φ, $\varphi(\overline{x})$ indicates that φ has all its free variables *among* $\{x_1,\ldots,x_n\}$, with $\varphi(\overline{T})$ denoting $\varphi[T_1/x_1,\ldots,T_n/x_n]$. Because variables are particular terms, we take the liberty to use the notation $\varphi(\overline{y})$ with two

different meanings, depending on the context: either to indicate that φ has its variables among $\{y_1, \ldots, y_n\}$, case in which $\varphi(\overline{y})$ is the same as φ, or to denote the formula obtained from φ by substituting the variables $\overline{x}$ (assumed indicated previously by writing φ as $\varphi(\overline{x})$) with the variables $\overline{y}$.

Let HORN^2 be the GFOL fragment given by the formulae:

$$\forall \overline{y}. \left(\forall \overline{x}. \bigwedge_{i=1}^{n} (a_i(\overline{x}, \overline{y}) \Rightarrow b_i(\overline{x}, \overline{y})) \right) \Rightarrow c(\overline{y}) \qquad (*)$$

where a_i, b_i, c are atomic formulae. We call these HORN^2*-formulae*. When a_i is $\top$ we write only $b_i(\overline{x}, \overline{y})$ instead of $a_i(\overline{x}, \overline{y}) \Rightarrow b_i(\overline{x}, \overline{y})$, and call the formula *extensional*; if in addition $\overline{x}$ has length 0, we obtain Horn formulae. When all b_i's are $\top$ or $n = 0$, we write $c(\overline{y})$ instead of $(*)$. HORN^2, extensional, and Horn sentences are by definition universal closures of corresponding types of formulae. We identify formulae with their universal closures in theories. A theory E is called HORN^2, extensional, or Horn if it consists of corresponding types of sentences. Besides including equational and Horn logics, HORN^2 can define any λ-calculus (untyped, typed, polymorphic, etc.) as shown in Section 3.

We shall eventually only consider Horn consequences (in other words, sequents $\Gamma \triangleright d$ with Γ a finite set of atomic formulae and d an atomic formula) of HORN^2 specifications, because only this type of consequences are usually relevant for λ-calculi. Moreover, all other HORN^2 consequences can be deduced from these using (generic forms of) the Constant Lemma and the Deduction Theorem. We first consider slightly more general sequents, namely ones of the form $\Gamma \triangleright \Delta$ with Γ and Δ finite sets of atomic formulae.

Fix a HORN^2 theory E. Our goal next is to simplify and specialize *with regard to* E the Gentzen system $\mathcal{G}$ discussed in the previous subsection. We first provide some immediate simplifications, based on the following remarks:

1. There is no need for the rules involving negation, because any provable negation-free sequent has a completed proof tree whose sequents do not contain negation;
2. There is no change in the strength of provability if we accept as axioms only those rules $\frac{\cdot}{\Gamma \triangleright \Delta}$ such that there exists an *atomic* sentence in $\Gamma \cap \Delta$ instead of *any* sentence, atomic or not; this is because whenever a sequent $\Gamma \triangleright \Delta$ is such that there is a compound formula φ in $\Gamma \cap \Delta$, then there is a non-axiom rule which can be applied backwards to it such that all its (one or two) upper sequents $\Gamma' \triangleright \Delta'$ have a strict subformula of φ in $\Gamma' \cap \Delta'$.

We obtain the following Gentzen system parameterized by E, denoted $\mathcal{G}'_E$, for entailing E-tautological sequents of the form $\Gamma \triangleright \Delta$, where Γ and Δ are finite sets of *atomic* formulae:

$$\frac{\cdot}{\Gamma \triangleright \Delta} \quad \text{if } \Gamma \cap \Delta \neq \emptyset \qquad \textsf{(Axiom)}$$

$$\frac{\Gamma a_i(\overline{z}, \overline{T}) \triangleright \Delta b_i(\overline{z}, \overline{T}) \text{ for } i = \overline{1,n}, \quad \Gamma c(\overline{T}) \triangleright \Delta}{\Gamma \triangleright \Delta} \qquad \textsf{(Inst'-}e\textsf{)}$$

In the rule (Inst'-e) above (the "instance of e" rule), e is a sentence in E of the form $(*)$ (thus a_i, b_i, c are the atomic formulae that build e), $\overline{z}$ is a *fresh* tuple of variables with the same length as $\overline{x}$, and $\overline{T}$ is a tuple of terms with the same length as $\overline{y}$. Here as well as in the other similar rules that we shall consider, we implicitly assume that if a_i is $\top$, then we do not add it to Γ, and if b_i is $\top$, we do not add the sequent $\Gamma a_i(\overline{z}, \overline{T}) \triangleright \Delta b_i(\overline{z}, \overline{T})$ to the hypotheses. Moreover, notice that if $n = 0$, the rule has only one hypothesis, $\Gamma c(\overline{T}) \triangleright \Delta$.

As opposed to $\mathcal{G}$, the system $\mathcal{G}'_E$ is parameterized by a fixed theory E; the axioms e in E do not appear in sequents as such, but lay on the background, yielding "instance" rules (Inst'-e). Therefore $\mathcal{G}'_E$, and also all the other Gentzen systems defined later in this subsection, are *specialized for (arbitrary but fixed) theories*. According to the discussion above, the following holds:

PROPOSITION 3. *The Gentzen system $\mathcal{G}'_E$ is (sound and) complete for deducing E-tautological sequents $\Gamma \triangleright \Delta$, where Γ and Δ are finite sets of atomic formulae.*

The rule (Inst'-e) can be split into a simpler instance rule (Inst-e) and a rule (Cut) as below:

$$\frac{\Gamma a_i(\overline{z}, \overline{T}) \triangleright \Delta b_i(\overline{z}, \overline{T}) \text{ for } i = \overline{1,n}}{\Gamma \triangleright \Delta c(\overline{T})} \qquad \textsf{(Inst-}e\textsf{)}$$

$$\frac{\Gamma \triangleright \Delta d, \quad \Gamma d \triangleright \Delta}{\Gamma \triangleright \Delta} \qquad \textsf{(Cut)}$$

Indeed, (Inst'-e) can immediately be simulated by (Inst-e) and (Cut); conversely, by completeness of $\mathcal{G}'_E$ and soundness of (Inst-e) and (Cut), any proof using (Inst-e) and (Cut) instead of (Inst'-e) can be performed in $\mathcal{G}'_E$. Let $\mathcal{G}^0_E$ denote the system consisting of (Axiom), (Inst-e) and (Cut). We thus obtained the following:

PROPOSITION 4. *The Gentzen system $\mathcal{G}^0_E$ is (sound and) complete for deducing E-tautological sequents $\Gamma \triangleright \Delta$, where Γ and Δ are finite sets of atomic formulae.*

Rules like the above (Cut) are usually undesirable for many reasons, among which their non-syntax-driven character due to the appearance of the interpolant d "out of nowhere". For us, this rule is undesirable because it increases the succedent of the sequents, thus not allowing one to only consider singleton antecedents leading to a simpler Gentzen system. Can (Cut) be always eliminated from $\mathcal{G}^0_E$? The answer is negative, as shown by the following "non-intuitionistic"[3] counterexample: E is $\{(a \Rightarrow b) \Rightarrow c, a \Rightarrow c\}$. Then $\emptyset \triangleright c$ is provable in $\mathcal{G}^0_E$: $\emptyset \triangleright c$ follows by (Cut) from $\emptyset \triangleright c\,a$ and $a \triangleright c$; $\emptyset \triangleright c\,a$ follows by (Inst-$((a \Rightarrow b) \Rightarrow c)$) from $a \triangleright a\,b$, the latter being an (Axiom); $a \triangleright c$ follows by (Inst-$(a \Rightarrow c)$) from $a \triangleright a$, the latter being an (Axiom). (Note that indeed c is a semantic consequence of E.) On the other hand, $\mathcal{G}^0_E$ without (Cut) cannot prove $\emptyset \triangleright c$, as the reader can easily see. However, some theories E allow for the elimination of (Cut), as shown below. Let $\mathcal{G}_E$ be $\mathcal{G}^0_E$ without (Cut), and $\mathcal{G}^1_E$ be $\mathcal{G}^0_E$ with (Cut) replaced by:

$$\frac{\Gamma \triangleright d, \quad \Gamma d \triangleright \Delta}{\Gamma \triangleright \Delta} \qquad \textsf{(Simple-Cut)}$$

Also consider the following family of rules:

$$\frac{\Gamma a(\overline{z}, \overline{T}) \triangleright \Delta}{\Gamma \triangleright \Delta} \qquad \textsf{(Drop-}(e, a)\textsf{)}$$

where e is a sentence in E of the form (*), $a(\overline{x}, \overline{y})$ is one of the a_i's, $\overline{z}$ is a fresh tuple of variables, and $\overline{T}$ is a tuple of terms of the same size as $\overline{y}$. We next prove that the simpler-to-check closure of $\mathcal{G}_E$ under the (Drop-(e, a)) rules ensures its closure under (Cut), hence allows for elimination of the latter rule from $\mathcal{G}^0_E$.

[3] Indeed, (Cut) could be eliminated if we considered an intuitionistic variant of GFOL; however, we do not get into this issue here.

LEMMA 1. *Assume that $\mathcal{G}_E$ is closed under the rules (Drop-(e,a)). If $\Gamma \rhd \Delta_1 \cup \Delta_2$ is derivable in $\mathcal{G}_E$, then either $\Gamma \rhd \Delta_1$ or $\Gamma \rhd \Delta_2$ is derivable in $\mathcal{G}_E$.*

LEMMA 2. *$\mathcal{G}_E$ and $\mathcal{G}^1_E$ are equivalent (i.e., (Simple-Cut) can be eliminated from $\mathcal{G}^1_E$).*

PROPOSITION 5. *If $\mathcal{G}_E$ is closed under the rules (Drop-(e,a)), then it is also closed under (Cut), i.e., then $\mathcal{G}_E$ is equivalent to $\mathcal{G}^0_E$.*

Finally, we are ready to prove the completeness of a particularly simple Gentzen system, for the case of certain HORN^2 theories. We let $\mathcal{K}_E$ denote the system obtained from $\mathcal{G}_E$ by restricting the succedents to be singletons, i.e., the following system:

$$\frac{}{\Gamma \rhd d} \quad \text{if } d \in \Gamma \qquad \text{(Axiom)}$$

$$\frac{\Gamma a_i(\overline{z},\overline{T}) \rhd b_i(\overline{z},\overline{T}) \text{ for } i = \overline{1,n}}{\Gamma \rhd c(\overline{T})} \qquad \text{(Inst-}e\text{)}$$

THEOREM 3. *If $\mathcal{G}_E$ is closed under the rules (Drop-(e,a)), then $\mathcal{K}_E$ is (sound and) complete for deducing E-tautological sequents $\Gamma \rhd d$, with Γ finite set of atomic formulae and d atomic formula.*

Theorem 3 extends seamlessly to cope with equality, since the equality axioms *Eql* are HORN^2 sentences themselves. (As mentioned before, the equality rules are obtained following the same "instanciation" technique from the axioms *Eql*.)

Let $\mathcal{G}^{=}_E$ and $\mathcal{K}^{=}_E$ denote the systems $\mathcal{G}_{(E \cup Eql)}$ and $\mathcal{K}_{(E \cup Eql)}$, regarded over the language with equality. We obtain the following, as an immediate consequence of Theorem 3:

THEOREM 4. *If $\mathcal{G}^{=}_E$ is closed under the rules (Drop-(e,a)),*[4] *then $\mathcal{K}^{=}_E$ is (sound and) complete for deducing, in the logic with equality, E-tautological sequents of the form $\Gamma \rhd d$, where Γ is a finite set of sentences and d is a sentence.*

We call a HORN^2 theory E *amenable* if it satisfies the hypothesis of Theorem 4 ($\mathcal{G}^{=}_E$ closed under (Drop-(e,a))).

Notice that, for the purpose of entailing sequents $\Gamma \rhd d$, $\mathcal{K}_E$ and $\mathcal{K}^{=}_E$ are highly optimized Gentzen systems – they function by directly applying the axioms of the theory E, and without the need of carrying over intermediate results in the succedents of the sequents. If E is a Horn theory, one obtains the well-known Hilbert system for Horn (and in particular, equational) logic: indeed, the antecedent being fixed in deductions, it can be omitted and thus the Gentzen system becomes a Hilbert system, writing $\frac{a}{b}$ for $\frac{\Gamma \rhd a}{\Gamma \rhd b}$ and keeping an implicit account for the effect of the (Axiom) rule, as any Hilbert system does. Thus, we obtained a *derivation* of the completeness result for the Horn logic w.r.t. to its simple Hilbert system from the one of FOL w.r.t. its more involved Gentzen system. More generally, if E is an extensional theory, then there are no (Drop-(e,a)) rules, hence E is also trivially amenable.

COROLLARY 1. *If E is an extensional theory, then $\mathcal{K}^{=}_E$ is complete for entailing E-tautological sequents of the form $\Gamma \rhd d$, where Γ is a finite set of sentences and d is a sentence.*

The above results will be relevant for our calculi specifications, as an amenable Horn^2 specification E of a calculus will recover, in the system $\mathcal{K}^{=}_E$, *the represented calculus itself* - see Section 3.2.

[4] In these rules, e ranges over the equality axioms as well.

3. Specifying Calculi in HORN^2

We here define several λ-calculi as HORN^2 theories; thus these calculi fall under HORN^2 in a rather direct way, just like group, field and vector-space theories fall under equational logic. Most of the calculi below were taken from [20, 15, 10]. For untyped λ-calculus, see [3]; System F was introduced in [8, 23], ML-style polymorphism in [14], and Edinburgh LF in [11]. A polymorphic calculus with units of measurement is studied in [12].

3.1 Specifications of Calculi

Terms below are considered up to their α-equivalence; substitution and free variables are standard, substitution acting on terms up to α-equivalence, in a *capture-avoiding* fashion – the term syntax axioms check easily in each case. We work in GFOL *with equality*.

We next recall some standard λ- and FOL- like notational conventions that we obey as well. Since our examples have two kinds of bindings (in terms and in formulae), we state these conventions explicitly, to avoid further confusion: **(1)** Both terms and formulae are considered up to α-equivalence. When an expression like $\lambda x.X$ appears in an axiom, λx is assumed to bind any occurrence of x in X (note that $\lambda x.\_$ is well-defined on α-equivalence classes, because $X \equiv_\alpha X'$ implies $\lambda x.X \equiv_\alpha \lambda x.X'$). **(2)** Term- and formula- binding operators are assumed to bind as far as they can: thus $\lambda x.x + x$ should be read as $\lambda x.(x + x)$ and $\forall x.\varphi \wedge \psi$ as $\forall x.(\varphi \wedge \psi)$; the conjunction ($\wedge$) binds stronger than the implication ($\Rightarrow$). **(3)** Formulae are identified with the sentences that are their *universal closures*, i.e., that quantify universally over all the free variables of the formulae. Therefore, a more rigorous way to write $x = \lambda y.x\,y$ and $(\lambda x.X)y = X[y/x]$ is $\forall x.x = \lambda y.x\,y$ and $\forall \overline{z}.(\lambda x.X)y = X[y/x]$ where $\overline{z}$ is the tuple of all variables free in $(\lambda x.X)y = X[y/x]$. **(4)** We use lower-case letters to denote variables and upper-case letters for terms. We call *metavariables* the symbols that we use to denote variables or terms (e.g., x, y, X, Y). Metavariables for variables and for terms are subject to different conventions. When two metavariables "x" and "y" appear in the same axiom, they denote some *fixed*, but *distinct* variables; therefore, e.g., $x = \lambda y.x\,y$ denotes a single GFOL sentence, with $x, y \in Var$ and x distinct from y (the choice of x and y is immaterial thanks to α-equivalence of formulae, since we assume an outer universal quantification, by convention 3). On the other hand, a metavariable "X" denotes an *arbitrary* term, thus, e.g., $(\lambda x.X)Y = X[Y/x]$ is an *axiom scheme* representing a set of formulae, one for each pair of terms (X, Y) (for more on axiom schemes see Appendix A); moreover, when X and Y appear in the same axiom scheme (like above), they are *not* assumed distinct.

When do we use variables and when terms? We use terms when there is no way to express the desired axiom as a single sentence. It turns out that axioms like the η-rule, classically stated as an *axiom scheme* ($E = \lambda y.E\,y$ with the side condition $y \notin FV(E)$), can be written as *a single formula* $x = \lambda y.x\,y$ in GFOL; indeed, the fact that x is "independent of y" is implicit in the way substitution is defined in a capture-avoiding fashion on (α-equivalencies of) terms. On the other hand, the β-rule cannot be written as a single formula, being inherently an axiom scheme.

Untyped λ-Calculus ($U\lambda$). An *unsorted* theory in HORN^2, with no relations except equality; terms are α-equivalence classes over syntax $Term ::= Var \mid Term\ Term \mid \lambda Var.Term$; axioms are:

$(\forall x.X = Y) \Rightarrow \lambda x.X = \lambda x.Y$	(ξ)
$(\lambda x.X)x = X$	(β)
$x = \lambda y.x\,y$	(η)

Remarks: (1) (ξ) is a proper extensional sentence scheme, i.e., it is *not* equivalent to one in a simpler fragment of GFOL, like Horn;

note that x is $\forall$-bound in the left of $\Rightarrow$, and λ-bound in the right, in order to achieve the desired meaning: if X and Y are equal for an arbitrary x, then they are equal as "functions" on x. Extensional sentences appear to suffice for untyped calculi.

(2) The very simple syntax of our (β)-rule may seem strange at first sight. However, it has a good underlying intuition: a function $\lambda x.X(x)$ applied to a *value* x yields $X(x)$ - in the latter "$X(x)$", the value x has substituted the variable, i.e., the *formal parameter* x. It is common practice to let the same symbol denote both the "formal" and the "actual" parameter. This brings no confusion, since in the equation $(\lambda x.X)x = X$ all occurrences of x in the left X are bound by λ, while the other occurrences of x are bound by the outer universal quantifier of the equation. Thus if we substitute a term Y for x in this equation we get the more conventional beta-rule $(\lambda x.X)Y = X[Y/x]$. One may replace the (β) axiom by any of its *GFOL-equivalent* forms $(\lambda x.X)y = X[y/x]$ and $(\lambda x.X)Y = X[Y/x]$; however, we find the current form of (β) quite elegant and compact.

(3) Non-extensional λ-calculus is obtained by removing (η).

(Simply-)Typed λ-Calculus ($T\lambda$). Sorts *type*, *data*. Relation symbol $typeOf : data \times type$. $Var = (Var_{type}, Var_{data}) = (TVar, DVar)$ and $Term = (Term_{type}, Term_{data}) = (TTerm, DTerm)$, where:

$TTerm ::= TVar \mid TConst \mid TTerm \to TTerm$

$DTerm ::= DVar \mid DTerm\ DTerm \mid \lambda\, DVar : TTerm.\ DTerm$

x, y, X, Y, t, t', and T, T' range over data variables, data terms, type variables, and type terms, respectively. This convention will apply to all specifications extending this one. For clarity, we enclose labels of typing formulae by square brackets (e.g., [App]), and labels of equational rules by parentheses (e.g., (β)).

$(\forall x.typeOf(x,t) \Rightarrow typeOf(X,t')) \Rightarrow typeOf(\lambda x{:}t.X, t \to t')$	[Abs]
$typeOf(x, t \to t') \wedge typeOf(y,t) \Rightarrow typeOf(x\,y, t')$	[App]
$(\forall x.typeOf(x,t) \Rightarrow X = Y) \Rightarrow \lambda x{:}t.X = \lambda x{:}t.Y$	(ξ)
$typeOf((\lambda x{:}t.X)x, t') \Rightarrow (\lambda x{:}t.X)x = X$	(β)
$typeOf(\lambda y{:}t.x\,y, t') \Rightarrow x = \lambda y{:}t.x\,y$	(η)

Remarks: (1) One usually also considers, besides the basic types such as *nat* and *bool* (which are elements of *TConst*), some data constants, such as 0, *succ*, $+$, or *ifThenElse*, with their assigned types (e.g., $typeOf(+, nat \to nat \to nat)$) and defining equations (e.g., $x{+}succ(y) = succ(x{+}y)$). To save space, we did not include these in our specification, nor we shall include such straightforward items in later specifications.

(2) The axioms [Abs] and (ξ) show that rules changing the typing context can be modelled using proper HORN^2 sentences, in a semantically clean manner; for instance, [Abs] says that we can type $\lambda x : t.X$ to $t \to t'$ whenever X has type t' for any value of its "argument" x of type t. Syntactically, this approach *simplifies the calculus*, since it allows one to focus on the actual *meaning* of axioms rather than on low-level details such as how to deal with typing contexts, free or fresh variables, etc.

(3) We allow type variables and quantify them in formulae, but this fact alone does *not* bring polymorphism, since we do not have abstraction over types. We can nevertheless use type variables to reason about types in general, which is not possible in simply-typed λ-calculus, so the HORN^2 specification above is slightly more powerful than simply-typed λ-calculus.

(4) Standard definitions of simply-typed λ-calculus make use of *typing contexts*. Our HORN^2 definition above does *not* make typing contexts explicit; they appear implicitly during the derivation process. A *typing judgement* of the form $x_1 : T_1, \ldots, x_n : T_n \triangleright X : T$ in the type-theoretic notation can be seen as syntactic sugar for $\forall x_1, \ldots, x_n.typeOf(x_1, T_1) \wedge typeOf(x_n, T_n) \Rightarrow typeOf(X, T)$. Because of the built-in equality axioms, we allow equations between data terms which are *not* necessarily well-typed. For example, $X = X$ holds regardless of whether X is well-typed. However, the conditions in equations make sure that we cannot deduce any equation $X = Y$, with X well-typed and Y non-well-typed. A typed equation of the form $x_1 : T_1, \ldots, x_n : T_n \triangleright X = Y : T$ can be seen as syntactic sugar for $(\forall x_1, \ldots, x_n.typeOf(x_1, T_1) \wedge typeOf(x_n, T_n)) \Rightarrow (typeOf(X, T) \wedge typeOf(Y, T) \wedge X = Y)$.

(5) Since compatibility of *typeOf* with equality is a built-in property of GFOL (and thus HORN^2), our specification enjoys the "type preservation" property (types are preserved by equalities) by default. That this property can be proved as a *theorem* in simply-typed λ-calculus ensures the correctness of our specification. In general, a property like type preservation is seen as a test for a calculus to be sound. If one wants to actually specify the calculus without this property and then prove it as a theorem, then one should use HORN^2 *without* equality and define "=" as an ordinary relation.

(6) Here and elsewhere, we state the typing conditions for equations as succinctly as possible; hence the above (β)-rule - its hypothesis, $typeOf((\lambda x : t.X)x, t')$ is usually split in two: $typeOf(\lambda x : t.X, t \to t') \wedge typeOf(x, t)$. The latter are equivalent with the former, via the [App]-rule, and in fact the compact form gives the essence of the hypothesis: the "problematic" term, $(\lambda x : t.X)x$, is well-typed. Moreover, we specify typing hypotheses for the equations only if needed, i.e., only if the equated terms are susceptible of "de-balancing" typing. Thus in the case of (β), X is surely well-typed whenever $(\lambda x : t.X)x$ is so; this way, with minimal precautions, we avoid allowing equalities between a well-typed term and a non-well-typed one. On the other hand, in the (ξ)-rule, provided $(\forall x.typeOf(x,t) \Rightarrow X = Y)$ holds, $\lambda x : t.X$ is well-typed iff $\lambda x : t.Y$ is so, hence there is no need for any typing hypotheses. This succinctness policy, with no spectacular results here, is useful for more complicated calculi, such as Type:Type (see below).

Typed λ-Calculus with Recursion ($T\lambda\mu$). Extends $T\lambda$.

$DTerm ::= \ldots \mid \mu\, DVar : TTerm.\ DTerm$

$(\forall x.typeOf(x,t) \Rightarrow typeOf(X,t)) \Rightarrow typeOf(\mu x{:}t.X, t)$	$[\mu]$
$typeOf(\mu x{:}t.X, t) \Rightarrow \mu x{:}t.X = X[\mu x{:}t.X/x]$	(μ)

System F ($\mathcal{SF}$). Extends $T\lambda$.

$TTerm ::= \ldots \mid \Pi\, TVar.\ TTerm$

$DTerm ::= \ldots \mid DTerm\ TTerm \mid \lambda\, TVar.\ DTerm$

$(\forall t.typeOf(X,T)) \Rightarrow typeOf(\lambda t.X, \Pi t.T)$	[T-Abs]
$typeOf(x, \Pi t.T) \Rightarrow typeOf(x\,t, T)$	[T-App]
$(\forall t.X = Y) \Rightarrow \lambda t.X = \lambda t.Y$	(Tξ)
$typeOf(\lambda t.X, t') \Rightarrow (\lambda t.X)t = X$	(Tβ)
$typeOf(\lambda t.x\,t, t') \Rightarrow x = \lambda t.x\,t$	(Tη)

Typed λ-Calculus with Subtyping ($T\lambda S$). Extends $T\lambda$. Adds a new relation symbol, $\leq$: $type \times type$.

$typeOf(x,t) \wedge t \leq t' \Rightarrow typeOf(x,t')$	[ST]
$t \leq t$	(Refl-ST)
$t \leq t' \wedge t' \leq t'' \Rightarrow t \leq t''$	(Trans-ST)
$t_1 \leq t_2 \wedge t_1' \leq t_2' \Rightarrow (t_2 \to t_1') \leq (t_1 \to t_2')$	(Arr-ST)

Typed λ-Calculus with Isorecursive Types ($T\lambda\mu T$). Extends $T\lambda$.

$TTerm ::= \ldots \mid \mu\, TVar.\ TTerm$

$DTerm ::= \ldots \mid fold\langle TTerm\rangle DTerm \mid unfold\langle TTerm\rangle DTerm$

$typeOf(x, T[(\mu t.T)/t]) \Rightarrow typeOf(fold\langle \mu t.T\rangle x, \mu t.T)$	[Fold]
$typeOf(x, \mu t.T) \Rightarrow typeOf(unfold\langle \mu t.T\rangle x, T[(\mu t.T)/t])$	[Unfold]
$unfold\langle \mu t.T\rangle(fold\langle \mu t.T\rangle x) = x$	(Inverse$_1$)
$fold\langle \mu t.T\rangle(unfold\langle \mu t.T\rangle x) = x$	(Inverse$_2$)

Subtyping Isorecursive Types. Extends both $T\lambda\mu T$ and $T\lambda S$.

$(\forall t, t'.(t \leq t') \Rightarrow (T \leq T')) \Rightarrow \mu t.T \leq \mu t'.T'$	(Amber)

Remark: The above axiom modeling the so-called "Amber rule" is another example of a proper HORN^2 formula, like the ones for (ξ) and typing of abstraction. Again, the rule makes perfect intuitive sense in this form with universally quantified hypothesis, that avoids considering any typing context.

Typed λ-Calculus with Type Operators and Binding ($T\lambda\omega$). Extends $T\lambda$ without [Abs] (which needs to be modified). Adds new sort, *kind*, and new relation $kindOf : type \times kind$. $Var = (Var_{kind}, Var_{type}, Var_{data}) = (KVar, TVar, DVar)$ and $Term = (Term_{kind}, Term_{type}, Term_{data}) = (KTerm, TTerm, DTerm)$, where:

$$\begin{array}{lcl} KTerm & ::= & * \mid KVar \mid KTerm \to KTerm \\ TTerm & ::= & \ldots \mid \lambda\, TVar : KTerm.\, TTerm \end{array}$$

k, k' range over kind variables.

$(kindOf(t,*) \wedge (\forall x.typeOf(x,t) \Rightarrow typeOf(X,t'))) \Rightarrow typeOf(\lambda x\!:\!t.X, t \to t')$	[Abs]
$kindOf(t,*) \wedge kindOf(t',*) \Rightarrow kindOf(t \to t',*)$	[K-Arr]
$(\forall t.kindOf(t,k) \Rightarrow kindOf(T,k')) \Rightarrow kindOf(\lambda t\!:\!k.T, k \to k')$	[K-Abs]
$kindOf(t, k \to k') \wedge kindOf(t', k) \Rightarrow kindOf(t\,t', k')$	[K-App]
$(\forall t.kindOf(t,k) \Rightarrow T = T') \Rightarrow \lambda t\!:\!k.T = \lambda t\!:\!k.T'$	(Kξ)
$kindOf((\lambda t\!:\!k.T)t, k') \Rightarrow (\lambda t\!:\!k.T)t = T$	(Kβ)
$typeOf(\lambda t\!:\!k.t'\,t, k') \Rightarrow t' = \lambda t\!:\!k.t'\,t$	(Kη)

Remark: Kinding is usually considered together with type polymorphism, to provide simple abbreviations such as $Pair = \lambda t.\lambda t'.\, \Pi t''.(t \to t' \to t'') \to t''$ as an alternative to parametric abbreviations such as $Pair\,t\,t' = \Pi t''.(t \to t' \to t'') \to t''$. In our logic, even the former abbreviations (usually kept in the meta-language - see [20]) can be stated at the logical level in a straightforward way: $\forall\, Pair.Pair = \lambda t.\lambda t'.\Pi t''.(t \to t' \to t'') \to t'' \Rightarrow \ldots$.

A polymorphic calculus with units of measurement ($T\lambda U$). Extends $T\lambda$. Adds a new sort, *unit*. $Var = (Var_{unit}, Var_{type}, Var_{data}) = (UVar, TVar, DVar)$ and $Term = (Term_{unit}, Term_{type}, Term_{data}) = (UTerm, TTerm, DTerm)$, where:

$$\begin{array}{lcl} UTerm & ::= & \ldots \mid UVar \mid 1 \mid UTerm \cdot UTerm \mid UTerm^{-1} \\ TTerm & ::= & \ldots \mid QType\ UTerm \mid \Pi\, UVar.\, TTerm \\ DTerm & ::= & \ldots \mid \lambda\, UVar.\, DTerm \mid DTerm\ UTerm \end{array}$$

u, u', u'' range over unit variables and U over unit terms.

$(\forall u.typeOf(X,T)) \Rightarrow typeOf(\lambda u.X, \Pi u.T)$	[U-Abs]
$typeOf(x, \Pi u.T) \Rightarrow typeOf(x\,u, T)$	[U-App]
$u \cdot u' = u' \cdot u$	(Comm)
$(u \cdot u') \cdot u'' = u \cdot (u' \cdot u'')$	(Assoc)
$u \cdot 1 = u$	(Id)
$u \cdot u^{-1} = 1$	(Inv)
$(\forall u.X = Y) \Rightarrow \lambda u.X = \lambda u.Y$	(U-ξ)
$typeOf((\lambda u.X)u, t') \Rightarrow (\lambda u.X)u = X$	(U-β)
$typeOf(\lambda u.x\,u, t') \Rightarrow x = \lambda u.x\,u$	(U-η)

Remarks: (1) To make the calculus meaningful, one also needs to consider basic units of measure, such as *kg* and m, as elements of *UTerm*.
(2) *QType* is a set of "quantitative" basic types, such as *nat* or *real*, for which it makes sense to consider units of measurement. Thus, for instance, $real\,m^2$ is the type of surfaces, while the type *real* should be seen as the polymorphic type $\Pi u.real\,u$ [12].

ML-Style Polymorphic λ-calculus ($ML\lambda$). Extends $U\lambda$; the imported sort is called *data*. Adds two sorts, *type* and *typeScheme*, with $type < typeScheme$ (thus we have an *order-sorted* setting) and two relations, $typeOf : data \times typeScheme$ and $moreGeneral : typeScheme \times typeScheme$. $Var = (Var_{typeScheme}, Var_{type}, Var_{data}) = (TSVar, TVar, DVar)$, with $TVar \subseteq TSVar$ and $Term = (Term_{typeScheme}, Term_{type}, Term_{data}) = (TSTerm, TTerm, DTerm)$, where:

$$\begin{array}{lcl} TSTerm & ::= & TSVar \mid TTerm \mid \Pi\, TVar.\, TSTerm \\ TTerm & ::= & TVar \mid TConst \mid TTerm \to TTerm \\ DTerm & ::= & \ldots \mid \text{let } DVar = DTerm \text{ in } DTerm \end{array}$$

x, y / X range over data variables/terms, t, t' / T over type variables/terms, and s, s', s'' / S over type-scheme variables/terms.

$typeOf(x,s) \wedge moreGeneral(s',s) \Rightarrow typeOf(x,s')$	[Inst]
$(\forall t.typeOf(x,S)) \Rightarrow typeOf(x, \Pi t.S)$	[Gen]
$(\forall x.typeOf(x,t) \Rightarrow typeOf(X,t')) \Rightarrow typeOf(\lambda x.X, t \to t')$	[Abs]
$typeOf(y,s) \wedge (\forall x.typeOf(x,s) \Rightarrow typeOf(X,s')) \Rightarrow typeOf(\text{let } x = y \text{ in } X, s')$	[Let]
$typeOf(x, t \to t') \wedge typeOf(y,t) \Rightarrow typeOf(x\,y, t')$	[App]
$typeOf(\text{let } x = y \text{ in } X,\ t) \Rightarrow \text{let } x = y \text{ in } X = X[y/x]$	(Let)
$moreGeneral(\Pi t.S, S)$	(MG$_1$)
$moreGeneral(s,s') \wedge moreGeneral(s',s'') \Rightarrow moreGeneral(s,s'')$	(MG$_2$)

Remarks: (1) The typing statement of the "let" construct uses type scheme variables, hence allows for polymorphism.
(2) The relation of being more general is defined in a very simple fashion; the rule (MG$_1$) says that the type scheme $\Pi t.S$ is more general that S with *any particular choice* for the type t appearing in S. Recall that the written formulae are meant to express their universal closures, in particular are meant to be universally quantified over t, hence any possible occurrence of t in the second S of $moreGeneral(\Pi t.S, S)$ is in the scope of an outer universal quantifier; this is precisely what "any particular choice" means. (See also the previous discussion on our β axiom.) **(3)** The typing rule [Gen] says that if we can associate the type (scheme) S to x for any type t, then we can regard the type S of x as polymorphic in t.

The Edinburgh LF Calculus with Dependent Types (LF). Sorts *kind*, *typeFamily*, *object*, and relations $typeOf : object \times typeFamily$ and $kindOf : typeFamily \times kind$. $Var = (Var_{kind}, Var_{typeFamily}, Var_{object}) = (KVar, TFVar, OVar)$ and $Term = (Term_{kind}, Term_{typeFamily}, Term_{object}) = (KTerm, TFTerm, OTerm)$, where:

$$\begin{array}{lcl} KTerm & ::= & KVar \mid type \mid \Pi_{OVar:TFTerm} KTerm \\ TFTerm & ::= & TFVar \mid \Pi_{OVar:TFTerm} TFTerm \\ & & \lambda\, OVar : TFTerm.TFTerm \mid TFTerm\ \ OTerm \\ OTerm & ::= & OVar \mid \lambda\, OVar : TFTerm.OTerm \mid OTerm\ OTerm \end{array}$$

x, y and X, Y range over object variables and terms, t, t' and T, T' over type-family variables and terms, and k, k' and K, K' over kind variables and terms.

$(\forall x.typeOf(x,t) \Rightarrow kindOf(T, type)) \Rightarrow kindOf(\Pi_{x:t}.T, type)$	[Pi-T]
$(\forall x.typeOf(x,t) \Rightarrow kindOf(T,K)) \Rightarrow kindOf(\lambda x\!:\!t.T, \Pi_{x:t}.K)$	[Abs-T]
$typeOf(x,t) \wedge kindOf(t', \Pi_{x:t}.K) \Rightarrow kindOf(t'\,x, K)$	[App-T]
$(\forall x.typeOf(x,t) \Rightarrow typeOf(X,T)) \Rightarrow typeOf(\lambda x\!:\!t.X, \Pi_{x:t}.T)$	[Abs-O]
$typeOf(x,t) \wedge typeOf(y, \Pi_{x:t}.T) \Rightarrow typeOf(y\,x, T)$	[App-O]
$kindOf((\lambda x\!:\!t.T)x, k) \Rightarrow (\lambda x\!:\!t.T)x = T$	(β-T)
$kindOf(\lambda x\!:\!t.t'x, k) \Rightarrow t' = \lambda x\!:\!t.t'x$	(η-T)
$typeOf((\lambda x\!:\!t.X)x, t') \Rightarrow (\lambda x\!:\!t.X)x = X$	(β-O)
$typeOf(\lambda x\!:\!t.y\,x, t') \Rightarrow y = \lambda x\!:\!t.y\,x$	(η-O)

Remarks: **(1)** As usual, we omit the signature consisting of type-family- and object- constants of various kinds and types.
(2) Notice again how the process of "instanciating" generic/formal parameters by concrete/actual parameters is handled. For instance, in the axiom [App-O] above, y has a type T "formally" dependent on x of type t, which means that when applied to an actual parameter of the required type, will yield something of type T with the formal parameter replaced by the actual one also referred to as x, but this time not covered by the Π-binding.
(3) Especially in intricate situations like this, where kinds, types, and data are combined in various ways, we claim that a HORN^2 definition of a calculus is more readable and intuitive than a type-context-based one.

Type:Type* λ*-Calculus *($TT\lambda$).* Unsorted. One relation, *typeOf*.
Term ::= *Var* | *type* | *Term Term*λ*Var* : *Term*.*Term* | Π*Var* : *Term*
x, t, t', u range over variables, and X, T over terms. We write x, X to refer what should be considered as data, t, t', T when types are meant, and u when the variable denotes either data or types. (These conventions are taken only for readability.)

$typeOf(type, type)$	[T:T]
$typeOf(t, type) \wedge (\forall u.typeOf(u,t) \Rightarrow typeOf(T, type)) \Rightarrow typeOf(\Pi u:t.T, type)$	[Π]
$typeOf(\Pi u:t.T, type) \wedge (\forall u.typeOf(u,t) \Rightarrow typeOf(X,T)) \Rightarrow typeOf(\lambda u:t.X, \Pi u:t.T)$	[Abs]
$(typeOf(\Pi u:t.T, type) \wedge typeOf(x, \Pi u:t.T) \wedge typeOf(u,t)) \Rightarrow typeOf(x\,u, T)$	[App]
$(\forall u.typeOf(u,t) \Rightarrow T = T') \Rightarrow \Pi u:t.T = \Pi u:t.T'$	($\xi\Pi$)
$(\forall u.typeOf(u,t) \Rightarrow X = Y) \Rightarrow \lambda u:t.X = \lambda u:t.Y$	($\xi\lambda$)
$typeOf((\lambda u:t.X)u, t') \Rightarrow (\lambda u:t.X)u = X$	(β)
$typeOf(\lambda u:t.x\,u, t') \Rightarrow x = \lambda u:t.x\,u$	(η)

3.2 Recovering the Original Calculi

The above HORN^2 specifications of calculi state axioms with a clear intuitive content. One can think of these axioms either *semantically*, as properties of the desired GFOL models, or *syntactically*, as constraints over the corresponding term syntax. But what is the precise relationship between our HORN^2 specifications and the traditional definitions of these calculi? While intuitively the relationship is very tight - they both follow the same intuitions about functions, typing, subtyping, etc. - this is obviously not precise enough. If we claim to have specified, or "defined", for instance, System F in HORN^2, we should be able to show that, indeed, our specification conforms System F. In other definitional frameworks, this is usually performed by a translation between the original system and its specification, which then needs to be shown *adequate*. In our case, it turns out that such a translation is *not necessary*, since the Gentzen system $\mathcal{K}^{=}_{E}$ (see Section 2.6) associated to the HORN^2 specification of the calculus has already done this.

For example, the next table lists all the "instance" rules given by the axioms of the HORN^2 specification $\mathcal{SF}$ of System F, i.e., all the rules of $\mathcal{K}^{=}_{\mathcal{SF}}$. For the sake of visual comparison with the traditional System-F definition, we shall use, in an infix style, the symbol ":" instead of "*typeOf*". Recall that lower-case letters like t and x, y denote type and data variables, and corresponding upper-case letters denote type and data terms. (Appendix A further clarifies why the system in the table below is indeed the Gentzen system $\mathcal{K}^{=}_{\mathcal{SF}}$ induced by the theory $\mathcal{SF}$.)

$\dfrac{\cdot}{\Gamma \triangleright x : T}$ if $(x:T)$ is in Γ	[Axiom]
$\dfrac{\Gamma\,(x:T) \triangleright X : T'}{\Gamma \triangleright (\lambda x{:}T.X) : T \to T'}$	[Inst-Abs]
$\dfrac{\Gamma \triangleright X : T}{\Gamma \triangleright (\lambda t.X) : \Pi t.T}$	[Inst-T-Abs]
$\dfrac{\Gamma \triangleright X : T \to T',\quad \Gamma \triangleright Y : T}{\Gamma \triangleright X\,Y : T'}$	[Inst-App]
$\dfrac{\Gamma \triangleright X : \Pi t.T}{\Gamma \triangleright X\,T' : T[t \leftarrow T']}$	[Inst-T-App]
$\dfrac{\cdot}{\Gamma \triangleright X = X}$	(Inst-Refl)
$\dfrac{\Gamma \triangleright X = Y}{\Gamma \triangleright Y = X}$	(Inst-Symm)
$\dfrac{\Gamma \triangleright X = Y,\quad \Gamma \triangleright Y = Z}{\Gamma \triangleright X = Z}$	(Inst-Trans)
$\dfrac{\cdot}{\Gamma \triangleright T = T}$	(Inst-T-Refl)
$\dfrac{\Gamma \triangleright T = T'}{\Gamma \triangleright T' = T}$	(Inst-T-Symm)
$\dfrac{\Gamma \triangleright T = T',\quad \Gamma \triangleright T' = T''}{\Gamma \triangleright T = T''}$	(Inst-T-Trans)
$\dfrac{\Gamma \triangleright X = Y,\quad \Gamma \triangleright T = T',\quad \Gamma \triangleright X : T}{\Gamma \triangleright Y : T'}$	(Inst-Comp.)
$\dfrac{\Gamma \triangleright X = Y}{\Gamma \triangleright Z[z \leftarrow X] = Z[z \leftarrow Y]}$	(Inst-Subst)
$\dfrac{\Gamma \triangleright T = T'}{\Gamma \triangleright T''[t \leftarrow T] = T''[t \leftarrow T']}$	(Inst-T-Subst)
$\dfrac{\Gamma\,(x:T) \triangleright X = Y}{\Gamma \triangleright \lambda x:T.X = \lambda x:T.Y}$	(Inst-ξ)
$\dfrac{\Gamma \triangleright X = Y}{\Gamma \triangleright \lambda t.X = \lambda t.Y}$	(Inst-T-ξ)
$\dfrac{\Gamma \triangleright (\lambda x:T.X)Y : T'}{\Gamma \triangleright (\lambda x:T.X)Y = X[x \leftarrow Y]}$	(Inst-β)
$\dfrac{\Gamma \triangleright (\lambda t.X)T : T'}{\Gamma \triangleright (\lambda t.X)T = X[t \leftarrow T]}$	(Inst-T-β)
$\dfrac{\Gamma \triangleright (\lambda y:T.Xy) : T'}{\Gamma \triangleright X = \lambda y:T.Xy}$	(Inst-η)
$\dfrac{\Gamma \triangleright (\lambda t.Xt) : T'}{\Gamma \triangleright X = \lambda t.Xt}$	(Inst-T-η)

The side conditions of the above HORN^2-Gentzen-system rules turn out to be the familiar ones: at [Inst-Abs] and (Inst-ξ), x does not occur (free) in Γ; at [Inst-T-Abs] and (Inst-T-ξ), t does not occur in Γ; at (Inst-η), $y \notin FV(X)$; at (Inst-T-η), $t \notin FV(X)$. Note that the equality rules are duplicated (and labelled (Inst-Refl), (Inst-T-Refl) etc.), since there are two sorts, *data* and *type*. We obtained indeed the System-F's original calculus, modulo a few minor adjustments discussed next.

If we pick any other specification from Section 3.1 and the corresponding original calculus, we shall discover a similar situation as in the case of System F. The Gentzen system underlying a HORN^2 theory that defines a calculus coincides with the original-calculus' definition, modulo the following three adjustments:

- Since we work in a logic with equality where relations such as typing are implicitly compatible with equality, type preservation holds by default; therefore, elimination of the (Inst-Comp.) rule above, i.e., a proof of type preservation, could be seen as an optimization of the HORN^2 Gentzen system. If one needs to define a calculus *without type preservation*, one clearly should

not use HORN^2 with equality, but could use HORN^2 without equality, treating "=" as an ordinary relation symbol.

- Because of the generality of the term syntax concept, one could not possibly know at the HORN^2 level how terms are actually built, e.g., using operations such as the application. Therefore, congruence rules like

$$\frac{\Gamma \rhd X = Y\ ,\ \ \Gamma \rhd X' = Y'}{\Gamma \rhd XX' = YY'} \quad \text{[Cong]}$$

are not available as tautological, i.e., built-in, rules in HORN^2. While such rules could indeed be enforced by axioms at the specification level, they are not necessary however, since their actual role would be to state compatibility with substitution, the latter being built in HORN^2 with equality - the (Inst-Subst) rule above. In conclusion, the Gentzen system for the HORN^2 theory corresponds to a substitution-based (rather than operation-congruence-based) version of the original calculus.[5]

- It is usually the case that some properties that are deducible from the HORN^2 definitions are not of much interest to the original calculus; for example, in the HORN^2 definition of System F, one could state and prove that $t = t$ for all types t. Moreover, due to the fact that here the antecedents Γ in sequents $\Gamma \rhd d$ may contain not only trivial typings $x : T$, associating data variables to type variables, but also more involved typings $X : T$ and equalities $X_1 = X_2$ or $T_1 = T_2$, one could also prove things like $(T_1 = T_3)\,(T_2 = T_3) \rhd (T_2 = T_3)$ or $(X : T)\,(T = T_1)\,(T_1 = T_2) \rhd X : T_2$. Note, however, that all these extra deducible properties are trivial – indeed, one can easily see that any combination of the rules (Inst-T-Refl), (Inst-T-Symm), (Inst-T-Trans), and (Inst-T-Subst$_:$) may only entail trivial equalities between types. Moreover, this extra pseudo-information *makes sense* for the original calculus itself (and thus it is *not* "junk"), just that the calculus does not bother to consider it - indeed, if one asks whether type equality is transitive or reflexive, the answer should be positive. For other calculi, such as the Type:Type λ-calculus and Edinburgh LF (our theories $TT\lambda$ and LF in Subsection 3.1) where one needs to perform more involved deductions with types, the fact that HORN^2 allows by default in its sentences variables ranging over all syntactic categories becomes convenient.

Therefore, a HORN^2 specification E brings in fact a *higher-level notation* for the calculus, because, by unfolding E into its afferent Gentzen system $\mathcal{K}^{=}_{E}$, one obtains the original calculus itself. Thus, one can think of $\mathcal{K}^{=}_{E}$ as the "traditional definition" of the calculus. However, $\mathcal{K}^{=}_{E}$ and E have *precisely* the same expressiveness only if E is amenable (Theorem 4). If E is an extensional theory, as for untyped λ-calculus, amenability and thus completeness of $\mathcal{K}^{=}_{E}$ come for free (Corollary 1); but for proper HORN^2 theories one needs to prove amenability. Thus, amenability corresponds to adequacy; note, however, a significant shift of focus: one needs *not* relate a HORN^2 theory to the "external" original definition of a calculus, as in the case of adequacy, but rather prove something only about the theory, namely, that its Gentzen system $\mathcal{G}^{=}_{E}$ is closed under the drop rules. Fortunately, like other proofs of drop-rule closures in λ-calculi (see, e.g., [15]), such proofs tend to be routine.

PROPOSITION 6. *All the theories E in Subsection 3.1 are amenable, and thus $\mathcal{K}^{=}_{E}$ is complete for any of them.*

We believe that any λ-calculus specification in HORN^2, if stated naturally, is amenable. Indeed, amenability means closure of $\mathcal{G}^{=}_{E}$ under the drop rules associated to the proper HORN^2 axioms of the specifications. For any proper HORN^2 axiom useful for defining calculi that we can imagine, its condition-of-the-condition states that a piece of data x is classified in a certain way, i.e., has a certain type or kind T, and thus its associated drop rule states that the information *typeOf*(x, T) or *kindOf*(x, T) with x completely fresh, i.e., completely unrelated to the context of the sequent, cannot help deduction. E.g., provided x is fresh for Γ and Δ, Γ*typeOf*$(x, T) \rhd \Delta$ is derivable only if $\Gamma \rhd \Delta$ is so. The information that "something" (referred to as x) has a type T (i.e., T is inhabited by x), does not reveal anything new in a context $\Gamma \rhd \Delta$ that is insensitive to "that something" (i.e., that does not contain x).

[5] Actually, most calculi definitions either use directly a substitution rule, or prove it as a derived rule.

4. Ad Hoc versus GFOL Models for λ-Calculi

GFOL provides models in a uniform manner to all its theories, in particular to all those in Section 3.1. We claim that this "general-purpose" GFOL semantics is as good/bad as "domain-specific" semantics previously proposed for some of these calculi. Not only it yields a notion of model for the particular calculus that makes deduction complete, but this notion resembles closely the domain-specific, ad hoc one. We exemplify this on untyped λ-calculus and on System F. For the former, the GFOL semantics *coincides* (up to a carrier-preserving bijection between classes of models) with its ad hoc, set-theoretical semantics from [3]; for the latter, GFOL provides a novel semantics, equivalent to the one given in [5]. Here equivalence means the existence of a bijection between elementary classes of models and will be brought by back and forth mappings between the classes of models, which preserve and reflect satisfaction; an equivalence brings an isomorphism between the *skeletons* of two logics [18] - this situation is similar to the one of equivalence of categories [13].

4.1 Untyped λ-Calculus

The term syntax of Untyped Lambda Calculus (ULC) was already mentioned in Section 3.1, for the specification $U\lambda$. We let Λ (instead of *Term*) denote the set of λ-terms over the countably infinite set *Var* of variables, modulo α-equivalence (we shall use the same notations as in [3]). In order to ease the presentation, we do not consider constants, but they could have been considered as well without any further difficulties. (This will be true for System F as well.) We recall from [3] some model-theoretic notions developed around ULC. Let us call *pre-structure* a triple $(A, \_\langle\_\rangle, (A_T)_{T\in\Lambda(A)})$, where A is a set, $\_\langle\_\rangle$ is a binary operation on A (i.e., $(A, \_\langle\_\rangle)$ is an applicative structure), and for each $T \in \Lambda(A)$, $A_T : A^{Var} \to A$, where $\Lambda(A)$ denotes the set of λ-terms with constants in A, modulo α-equivalence.

Given an equation $T_1 = T_2$ with T_1, T_2 λ-terms, one defines $A \models_\lambda T_1 = T_2$ as usual, by interpreting $T_1 = T_2$ as being implicitly universally quantified - that is, by $A_{T_1}(\rho) = A_{T_2}(\rho)$ for all $\rho : Var \to A$. For pre-structures, we consider the following properties (where a, b range over elements of A, x over variables, T, T_1, T_2 over terms, ρ, ρ' over valuations, i.e., elements of A^{Var}):

(P1) $A_x(\rho) = \rho(x)$;

(P2) $A_{T_1T_2}(\rho) = A_{T_1}(\rho)\langle A_{T_2}(\rho)\rangle$;

(P3) If $\rho{\restriction}_{FV(T)} = \rho'{\restriction}_{FV(T)}$, then $A_T(\rho) = A_T(\rho')$;

(P4) If $A_T(\rho[x \leftarrow a]) = A_{T'}(\rho[x \leftarrow a])$ for all $a \in A$, then $A_{\lambda x.T}(\rho) = A_{\lambda x.T'}(\rho)$;

(P5) $A_{\lambda x.T}(\rho)\langle a\rangle = A_T(\rho[x \leftarrow a])$;

(P6) If $a\langle c\rangle = b\langle c\rangle$ for all $c \in A$, then $a = b$;

(P7) $A_a(\rho) = a$.

We next simplify the pre-structures slightly, by removing their redundant data given by parameterized terms. A *simple pre-structure* is a triple $(A, \_\langle\_\rangle, (A_T)_{T\in\Lambda})$ which satisfies all properties of a pre-structure, except (P7). Note that the difference between simple pre-structures and pre-structures is that only terms in Λ, and not in $\Lambda(A)$, are considered. Hence the notion of satisfaction, defined for pre-structures only w.r.t. equations involving terms in Λ, also makes sense for simple pre-structures. We shall only be interested in pre-structures verifying at least (P1)-(P4). In this case, simple pre-structures and pre-structures are essentially identical:

LEMMA 3. *The forgetful mapping $(A, \_\langle\_\rangle, (A_T)_{T\in\Lambda(A)})$ to $(A, \_\langle\_\rangle, (A_T)_{T\in\Lambda})$ is a bijection, preserving satisfaction and each of the properties (P5),(P6), between pre-structures verifying (P1)-(P4) and (P7) and simple pre-structures verifying (P1)-(P4).*

This lemma allows us to work with the more amenable simple pre-structures, which we henceforth call "pre-structures", and forget about the more complicated ones, as well as about property (P7).

A *syntactical λ-model* [3] (*λ-model* for short) is a pre-structure verifying (P1)-(P5). A λ-model is *extensional* if it verifies (P6).

We now come to the representation of ULC in HORN^2. Consider the generic first-order language $(Var, \Lambda, \emptyset)$, whose models have therefore the form $(A, (A_T)_{T\in\Lambda})$. Rather than considering only the HORN^2 theory $U\lambda$, we prefer to play around with more combinations of HORN^2 formulae (among which $U\lambda$), in order to allow a closer look at the relationship between the two types of models. We shall work with the following HORN^2 formulae and schemes of formulae:[6]

$(\lambda x.T)T' = T[T'/x]$	(β)
$(\lambda x.T)x = T$	(β')
$\lambda x.T\, x = T$, if $x \notin FV(T)$	(η)
$\lambda x.y\, x = y$	(η')
$(\forall x.T_1 = T_2) \Rightarrow \lambda x.T_1 = \lambda x.T_2$	(ξ)
$(\forall x.T_1\, x = T_2\, x) \Rightarrow T_1 = T_2$, if $x \notin FV(T)$	(ext)
$(\forall x.y_1 x = y_2 x) \Rightarrow y_1 = y_2$	(ext')

Note that we do not use the same notations as in Section 3.1. $U\lambda$ here consists of (β'), (ξ), and (η').

LEMMA 4. *Each of the schemes of formulae (β), (η), (ext) is semantically equivalent in GFOL to its primed variant.*

This lemma points out that side-conditioned axiom schemes like (η) and (ext) are *not necessary* in GFOL, since they are expressible as single sentences, (η') and (ext').

We define a correspondence between pre-structures verifying (P1)-(P4) and GFOL models satisfying ξ as follows:

- Each pre-structure verifying (P1)-(P4) $L = (A, \_\langle\_\rangle, (A_T)_{T\in\Lambda})$ is mapped to a GFOL model $L^{\#} = (A, (A_T)_{T\in\Lambda})$;
- Each GFOL model $M = (A, (A_T)_{T\in\Lambda})$ satisfying (ξ) is mapped to a pre-structure $M^{\$} = (A, \_\langle\_\rangle, (A_T)_{T\in\Lambda})$, where $\_\langle\_\rangle$ is defined by $a\langle b\rangle = A_{xy}(\rho)$, with ρ taking x to a and y to b.

PROPOSITION 7. *The above two mappings are well defined and mutually inverse. Moreover, they preserve satisfaction and they can be restricted and corestricted to:*

(a) λ-models versus GFOL models satisfying $(\xi), (\beta)$ (i.e., models of $U\lambda$ without (η));
(b) extensional λ-models versus GFOL models satisfying $(\xi), (\beta), (\eta)$ (i.e., models of $U\lambda$).

[6] Recall the conventions regarding axioms and axiom schemes.

Considering the above satisfaction-preserving bijections, one could say that the syntactic models *coincide* with the GFOL models for the corresponding theories.

4.2 System F

The syntactic categories of System F [8, 23] are defined in Section 3.1 (as the two-sorted term syntax of theory $\mathcal{SF}$). These coincide with those of System F as defined in the literature, just that we call "data terms" and "type terms" what are traditionally referred to as "terms" and "types". A *typing context* is a finite set $\{x_1 : T_1, \ldots, x_n : T_n\}$ where x_i's are data variables, T_i's are type terms, and no data variable appears twice. Below x, y and X, Y range over data variables and terms, t, t' and T, T' over type variables and terms, and Γ over typing contexts. The typing system for System F derives *typing judgements*, i.e., triples $\Gamma \rhd X : T$, and is given by the following rules:

$\dfrac{\cdot}{\Gamma \rhd x : T}$ if $(x:T) \in \Gamma$	[SF-InVar]
$\dfrac{\Gamma\,(x:T) \rhd X : T'}{\Gamma \rhd (\lambda x{:}T.X) : T \to T'}$	[SF-Abs]
$\dfrac{\Gamma \rhd X : T}{\Gamma \rhd (\lambda t.X) : \Pi t.T}$	[SF-T-Abs]
$\dfrac{\Gamma \rhd X : T \to T' \quad \Gamma \rhd Y : T}{\Gamma \rhd X\, Y : T'}$	[SF-App]
$\dfrac{\Gamma \rhd X : \Pi t.T}{\Gamma \rhd X\, T' : T[T'/t]}$	[SF-T-App]

At the rules [SF-AddVar] and [SF-Abs], $\Gamma\,(x:T)$ [7] is assumed to be a typing context with $(x:T) \notin \Gamma$, i.e., it is assumed that x is not free in the left of any pair in Γ. At [SF-T-Abs], it is assumed that t is not free in the right of any pair in Γ. We let $\vdash_{SF}\ \Gamma \rhd X : T$ denote the fact that $\Gamma \rhd X : T$ is deducible in the above system.

We relate System F and the HORN^2 theory $\mathcal{SF}$ first w.r.t. typing. For each typing context $\Gamma = \{x_1 : T_1, \ldots, x_n : T_n\}$, we let $\Gamma^{\#}$ be the GFOL formula $\mathit{typeOf}(x_1, T_1) \wedge \ldots \wedge \mathit{typeOf}(x_n, T_n)$.

PROPOSITION 8. *For all typing judgements $\Gamma \rhd X : T$, $\vdash_{SF} \Gamma \rhd X : T$ iff $\mathcal{SF} \vdash_{GFOL} \forall FV(X,T).\Gamma^{\#} \Rightarrow \mathit{typeOf}(X,T)$.*

A *Henkin model* H for System F [5, 15] is a tuple $(\mathcal{T}, \mathcal{F}, \to, \Pi, \mathcal{I}_{type}, (\mathcal{D}om_\tau)_{\tau\in\mathcal{T}}, (App_{\tau,\sigma})_{\tau,\sigma\in\mathcal{T}}, (App_f)_{f\in\mathcal{F}}, \mathcal{I})$, together with a pair $((H_T)_{T\in TTerm}, (H_{tj})_{tj\in Tj, \vdash tj})$, where:
(a) $\to : \mathcal{T} \times \mathcal{T} \to \mathcal{T}$,
(b) $\Pi : \mathcal{F} \to \mathcal{T}$,
(c) $\mathcal{F} \subseteq \mathcal{T}^{\mathcal{T}}$,
(d) $\mathcal{I}_{type} : TConst \to \mathcal{T}$,
(e) $App_{\tau,\sigma} : \mathcal{D}om_{\tau\to\sigma} \to \mathcal{D}om_\sigma{}^{\mathcal{D}om_\tau}$ for each $\tau, \sigma \in \mathcal{T}$,
(f) $App_f : \mathcal{D}om_{\Pi f} \to \prod_{\tau\in\mathcal{T}} \mathcal{D}om_{f(\tau)}$ for each $f \in \mathcal{F}$,
(g) $H_T : \mathcal{T}^{TVar} \to \mathcal{T}$ for each $T \in TTerm$,
(h) $H_{\Gamma\rhd X:T} : \{(\gamma, \delta) \in \mathcal{T}^{TVar} \times (\bigcup \mathcal{D}om)^{DVar} : \text{for all } x : T' \in \Gamma,\ \delta(x) \in \mathcal{D}om_{H_{T'}(\gamma)}\} \to \bigcup \mathcal{D}om$ for each $\Gamma \rhd X : T \in Tj$ with $\vdash_{SF} \Gamma \rhd X : T$, such that the following hold:
(1) Each of $App_{\tau,\sigma}$ and App_f is injective;
(2) $(\tau \mapsto H_T(\gamma[t \leftarrow \tau])) \in \mathcal{F}$ for each T, t, γ;
(3) $H_t(\gamma) = \gamma(t)$ for each $t \in TVar$;
(4) $H_{tc}(\gamma) = \mathcal{I}_{type}(tc)$ for each $tc \in TConst$;
(5) $H_{T\to T'}(\gamma) = H_T(\gamma) \to H_{T'}(\gamma)$;
(6) $H_{\Pi t.T}(\gamma) = \Pi(\tau \mapsto H_T(\gamma[t \leftarrow \tau]))$;

[7] We keep a convention similar to the one of Section 2.5, that $\Gamma\,(x:T)$ is a notation for $\Gamma \cup \{(x:T)\}$.

(7) $H_{\Gamma \triangleright x:T}(\gamma,\delta) = \delta(x)$;
(8) $H_{\Gamma \triangleright XY:T}(\gamma,\delta) =$
$App_{H_{T'}(\gamma),H_T(\gamma)}(H_{\Gamma \triangleright X:T' \to T}(\gamma,\delta))(H_{\Gamma \triangleright Y:T'}(\gamma,\delta))$;
(9) $H_{\Gamma \triangleright XT:T'[T/t]}(\gamma,\delta) =$
$App_{\tau \mapsto H_{T'}(\gamma[t \leftarrow \tau])}(H_{\Gamma \triangleright X:\Pi t.T'}(\gamma,\delta))(H_T(\gamma))$;

(10) $H_{\Gamma \triangleright \lambda x:T.X:T \to T'}(\gamma,\delta) \in \mathcal{D}om_{H_T(\gamma) \to H_{T'}(\gamma)}$ and, for each $d \in \mathcal{D}om_{H_T(\gamma)}$, $App_{H_T(\gamma),H_{T'}(\gamma)}(H_{\Gamma \triangleright \lambda x:T.X:T \to T'}(\gamma,\delta))(d) = H_{\Gamma \cup \{x:T\} \triangleright X:T'}(\gamma, \delta[x \leftarrow d])$;
(11) $H_{\Gamma \triangleright \lambda t.X:\Pi t.T}(\gamma,\delta) \in$
$\mathcal{D}om_{\Pi(\tau \mapsto H_T(\gamma[t \leftarrow \tau]))}$ and, for each $\tau \in \mathcal{T}$, $App_{\tau \mapsto H_T(\gamma[t \leftarrow \tau])}$ $(H_{\Gamma \triangleright \lambda t.X:(\Pi t)T}(\gamma,\delta))(\tau) = H_{\Gamma \triangleright X:T}(\gamma[t \leftarrow \tau], \delta)$.

Above, *Tj* denotes the set of typing judgements and $\bigcup \mathcal{D}om$ denotes $\bigcup_{\tau \in \mathcal{T}} \mathcal{D}om_{f(\tau)}$; *tj* ranges over typing judgements, σ, τ and d, d' over elements of $\mathcal{T}$ and $\bigcup \mathcal{D}om$, γ and δ over maps in $\mathcal{T}^{TVar}$ and $(\bigcup \mathcal{D}om)^{DVar}$; $\tau \mapsto H_T(\gamma[t \leftarrow \tau])$ denotes the function mapping each τ to $H_T(\gamma[t \leftarrow \tau])$. We use slightly different notations than [15]; also, we include interpretations of types and well-typed terms H_T and H_{tj} as part of the structure, while [15] equivalently asks that such interpretations exist and then proves them unique.

Satisfaction by Henkin models H of well-typed equations $\Gamma \triangleright X = Y : T$ (with $\vdash_{SF} \Gamma \triangleright X : T$ and $\vdash_{SF} \Gamma \triangleright X : T$) is defined by $H \models_{SF} \Gamma \triangleright X = Y : T$ iff $H_{\Gamma \triangleright X:T} = H_{\Gamma \triangleright Y:T}$.

To avoid technical details irrelevant here, we assume non-emptiness of types (without such an assumption, the Henkin models are *not* complete for System F, but only if one considers a richer language - see [15]). Next we define mappings between System-F Henkin models and GFOL models for $\mathcal{SF}$. Given three sets A, B, C, a mapping $f : A \times B \to C$ is called *extensional* if for all $a, a' \in A$, if $f(a,b) = f(a',b)$ for all $b \in B$ then $a = a'$. Below, the satisfaction relation for 1-,2-,3-,4-, and 5- Henkin models is defined similarly to that for Henkin models.

The first transformations are:

- Consider each $App_{\tau,\sigma}$ not as an injective mapping $\mathcal{D}om_{\tau \to \sigma} \to \mathcal{D}om_\sigma{}^{\mathcal{D}om_\tau}$, but as an extensional mapping $\mathcal{D}om_{\tau \to \sigma} \times \mathcal{D}om_\tau \to \mathcal{D}om_\sigma$;
- Consider each App_f not as an injective mapping $\mathcal{D}om_{\Pi f} \to \prod_{\tau \in \mathcal{T}} \mathcal{D}om_{f(\tau)}$, but as an extensional mapping $\mathcal{D}om_{\Pi f} \times \mathcal{T} \to \bigcup \mathcal{D}om$ such that for each $(d,\tau) \in \mathcal{D}om_{\Pi f} \times \mathcal{T}$, $App_f(d,\tau) \in \mathcal{D}om_{f(t)}$;
- Assume $\mathcal{F}$ consists only of mappings of the form $\tau \mapsto H_T(\gamma[t \leftarrow \tau])$ for some $T \in$ *TTerm*, $t \in$ *TVar* and $\gamma \in \mathcal{T}^{TVar}$; only this kind of mappings are used in the Henkin model definition, and thus in the definition of satisfaction;
- Assume all $\mathcal{D}om_\tau$ and $\mathcal{D}om_\sigma$, with $\tau \neq \sigma$, mutually disjoint; this obviously does not affect the satisfaction relation.

We thus obtain the following equivalent models for System F:

A *1-Henkin model* H is a a tuple
$(\mathcal{T}, \mathcal{F}, \to, \Pi, \mathcal{I}_{type}, (\mathcal{D}om_\tau)_{\tau \in \mathcal{T}}, (App_{\tau,\sigma})_{\tau,\sigma \in \mathcal{T}}, (App_f)_{f \in \mathcal{F}}, \mathcal{I})$,
together with a pair $((H_T)_{T \in TTerm}, (H_{tj})_{tj \in Tj, \vdash tj})$, where:
(a) $\to$: $\mathcal{T} \times \mathcal{T} \to \mathcal{T}$,
(b) $\Pi : \mathcal{F} \to \mathcal{T}$,
(c) $\mathcal{F} \subseteq \mathcal{T}^{\mathcal{T}}$, $\mathcal{F} = \{\tau \mapsto H_T(\gamma[t \leftarrow \tau]) : T \in TTerm, t \in TVar, \gamma \in \mathcal{T}^{TVar}\}$,
(d) $\mathcal{I}_{type} : TConst \to \mathcal{T}$,
(e) $App_{\tau,\sigma} : \mathcal{D}om_{\tau \to \sigma} \times \mathcal{D}om_\tau \to \mathcal{D}om_\sigma$ for each $\tau, \sigma \in \mathcal{T}$,
(f) $App_f : \mathcal{D}om_{\Pi f} \times \mathcal{T} \to \bigcup \mathcal{D}om$ for each $f \in \mathcal{F}$,
(g) $H_T : \mathcal{T}^{TVar} \to \mathcal{T}$ for each $T \in TTerm$,
(h) $H_{\Gamma \triangleright X:T} : \{(\gamma,\delta) \in \mathcal{T}^{TVar} \times (\bigcup \mathcal{D}om)^{DVar}$: for all $x : T' \in \Gamma$, $\delta(x) \in \mathcal{D}om_{H_{T'}(\gamma)}\} \to \bigcup \mathcal{D}om$ for each $\Gamma \triangleright X : T \in Tj$ with $\vdash_{SF} \Gamma \triangleright X : T$,
such that the following hold:
(1) $\mathcal{D}om_\tau \cap \mathcal{D}om_\sigma = \emptyset$ whenever $\tau \neq \sigma$; $dc \in DConst$;
(2) Each of $App_{\tau,\sigma}$ and App_f is extensional, and $App_f(d,\tau) \in \mathcal{D}om_{f(t)}$ for all $(d,\tau) \in \mathcal{D}om_{\Pi f} \times \mathcal{T}$;
(3) $H_t(\gamma) = \gamma(t)$ for each $t \in TVar$;
(4) $H_{tc}(\gamma) = \mathcal{I}_{type}(tc)$ for each $tc \in TConst$;
(5) $H_{T \to T'}(\gamma) = H_T(\gamma) \to H_{T'}(\gamma)$;
(6) $H_{\Pi t.T}(\gamma) = \Pi(\tau \mapsto H_T(\gamma[t \leftarrow \tau]))$;
(7) $H_{\Gamma \triangleright x:T}(\gamma,\delta) = \delta(x)$
(8) $H_{\Gamma \triangleright XY:T}(\gamma,\delta) =$
$App_{H_{T'}(\gamma),H_T(\gamma)}(H_{\Gamma \triangleright X:T' \to T}(\gamma,\delta),\ H_{\Gamma \triangleright Y:T'}(\gamma,\delta))$;
(9) $H_{\Gamma \triangleright XT:T'[T/t]}(\gamma,\delta) =$
$App_{\tau \mapsto H_{T'}(\gamma[t \leftarrow \tau])}(H_{\Gamma \triangleright X:\Pi t.T'}(\gamma,\delta),\ H_T(\gamma))$;
(10) $H_{\Gamma \triangleright \lambda x:T.X:T \to T'}(\gamma,\delta) \in \mathcal{D}om_{H_T(\gamma) \to H_{T'}(\gamma)}$ and, for each $d \in \mathcal{D}om_{H_T(\gamma)}$, $App_{H_T(\gamma),H_{T'}(\gamma)}(H_{\Gamma \triangleright \lambda x:T.X:T \to T'}(\gamma,\delta),\ d) = H_{\Gamma \cup \{x:T\} \triangleright X:T'}(\gamma, \delta[x \leftarrow d])$;
(11) $H_{\Gamma \triangleright \lambda t.X:\Pi t.T}(\gamma,\delta) \in \mathcal{D}om_{\Pi(\tau \mapsto H_T(\gamma[t \leftarrow \tau]))}$ and, for each $\tau \in \mathcal{T}$, $App_{\tau \mapsto H_T(\gamma[t \leftarrow \tau])}(H_{\Gamma \triangleright \lambda t.X:(\Pi t)T}(\gamma,\delta),\ \tau) =$
$H_{\Gamma \triangleright X:T}(\gamma[t \leftarrow \tau], \delta)$.

LEMMA 5. *Henkin and 1-Henkin models are equivalent, in that there exists a satisfaction- preserving and reflecting surjection* $\&_1$ *between the class of Henkin models and that of 1-Henkin models. That is to say: for all Henkin models* H *and well-typed equation* $\Gamma \triangleright X = Y : T$,

$$H \models \Gamma \triangleright X = Y : T \quad \textit{iff} \quad H^{\&_1} \models \Gamma \triangleright X = Y : T.$$

For the next modification we do not introduce a new model name. We simply assume that the 1-Henkin models have the mappings Π and $\to$ injective. It is conceptually straightforward that by taking this assumption we obtain equivalent models. Indeed, any 1-Henkin model H with non-injective Π and $\to$ can be transformed, without affecting the satisfaction relation, into one with injective Π and $\to$, by tagging the results of these mappings applications with the arguments.

We next simplify the 1-Henkin models by getting rid of their functional component $\mathcal{F}$. We base this simplification on the fact that, by the injectivity of Π, we can replace the index f of *App* with $H_{\Pi t.T}(\gamma)$, where $f \in \mathcal{F}$ has the form $\tau \mapsto H_T(\gamma[\tau \leftarrow t])$.

A *2-Henkin model* H is a a tuple
$(\mathcal{T}, \to, \mathcal{I}_{type}, (\mathcal{D}om_\tau)_{\tau \in \mathcal{T}}, (App_{\tau,\sigma})_{\tau,\sigma \in \mathcal{T}}, (App_\tau)_{\tau \in \mathcal{T}}, \mathcal{I})$, together with a pair $((H_T)_{T \in TTerm}, (H_{tj})_{tj \in Tj, \vdash tj})$, where:
(a) $\to$: $\mathcal{T} \times \mathcal{T} \to \mathcal{T}$ is an injective mapping,
(b) $\mathcal{I}_{type} : TConst \to \mathcal{T}$,
(c) $App_{\tau,\sigma} : \mathcal{D}om_{\tau \to \sigma} \times \mathcal{D}om_\tau \to \mathcal{D}om_\sigma$ for each $\tau, \sigma \in \mathcal{T}$,
(d) $App_{H_{\Pi t.T}(\gamma)} : \mathcal{D}om_{H_{\Pi t.T}(\gamma)} \times \mathcal{T} \to \bigcup \mathcal{D}om$ for each γ, t, T,
(e) $H_T : \mathcal{T}^{TVar} \to \mathcal{T}$ for each $T \in TTerm$,
(f) $H_{\Gamma \triangleright X:T} : \{(\gamma,\delta) \in \mathcal{T}^{TVar} \times (\bigcup \mathcal{D}om)^{DVar}$: for all $x : T' \in \Gamma$, $\delta(x) \in \mathcal{D}om_{H_{T'}(\gamma)}\} \to \bigcup \mathcal{D}om$ for each $\Gamma \triangleright X : T \in Tj$ with $\vdash_{SF} \Gamma \triangleright X : T$,
such that the following hold:
(1) $\mathcal{D}om_\tau \cap \mathcal{D}om_\sigma = \emptyset$ whenever $\tau \neq \sigma$;
(2) Each of $App_{\tau,\sigma}$ and $App_{H_{\Pi t.T}(\gamma)}$ is extensional, and $App_{H_{\Pi t.T}(\gamma)}(d,\tau) \in \mathcal{D}om_{H_T(\gamma[t \leftarrow \tau])}$ for each t, T, γ and $(d,\tau) \in \mathcal{D}om_{H_{\Pi t.T}(\gamma)} \times \mathcal{T}$;
(3) $H_t(\gamma) = \gamma(t)$ for each $t \in TVar$;
(4) $H_{tc}(\gamma) = \mathcal{I}_{type}(tc)$ for each $tc \in TConst$;
(5) $H_{T \to T'}(\gamma) = H_T(\gamma) \to H_{T'}(\gamma)$;
(6) $H_{\Gamma \triangleright x:T}(\gamma,\delta) = \delta(x)$

(7) $H_{\Gamma \rhd XY:T}(\gamma, \delta) =$
$App_{H_{T'}(\gamma), H_T(\gamma)}(H_{\Gamma \rhd X:T' \to T}(\gamma, \delta),\ H_{\Gamma \rhd Y:T'}(\gamma, \delta))$;
(8) $H_{\Gamma \rhd XT:T'[T/t]}(\gamma, \delta) =$
$App_{H_{\Pi t.T'}(\gamma)}(H_{\Gamma \rhd X:\Pi t.T'}(\gamma, \delta),\ H_T(\gamma))$;
(9) $H_{\Gamma \rhd \lambda x:T.X:T \to T'}(\gamma, \delta) \in \mathcal{D}om_{H_T(\gamma) \to H_{T'}(\gamma)}$ and, for each $d \in \mathcal{D}om_{H_T(\gamma)}$, $App_{H_T(\gamma), H_{T'}(\gamma)}(H_{\Gamma \rhd \lambda x:T.X:T \to T'}(\gamma, \delta),\ d) = H_{\Gamma \cup \{x:T\} \rhd X:T'}(\gamma, \delta[x \leftarrow d])$;
(10) $H_{\Gamma \rhd \lambda t.X:\Pi t.T}(\gamma, \delta) \in \mathcal{D}om_{H_{\Pi t.T}(\gamma)}$ and, for each $\tau \in \mathcal{T}$, $App_{H_{\Pi t.T}(\gamma)}(H_{\Gamma \rhd \lambda t.X:(\Pi t)T}(\gamma, \delta),\ \tau) = H_{\Gamma \rhd X:T}(\gamma[t \leftarrow \tau], \delta)$.

LEMMA 6. *1-Henkin and 2-Henkin models are equivalent, in that there exist two satisfaction- preserving and reflecting mappings $\&_2^1$ and $\&_1^2$ between the two classes of models.*

Next we flatten the multi-typed domain $\mathcal{D}om$ of 2-Henkin models. The flattening is based on the following:

- The multi-typing of $\mathcal{D}om$ can be viewed as a relation *typeOf* between data and types;
- Due to the type-wise disjointness of the domain and the injectivity of $\to$, the families of mappings $(App_{\tau,\sigma})_{\tau,\sigma \in \mathcal{T}}$ and $(App_\tau)_{\tau \in \mathcal{T}}$ can be replaced by two mappings $App : \bigcup \mathcal{D}om \times \bigcup \mathcal{D}om \to \bigcup \mathcal{D}om$ and $TApp : \bigcup \mathcal{D}om \times \mathcal{T} \to \bigcup \mathcal{D}om$, with postulating the necessary typing restrictions; since *App* and *TApp* will be total functions, we allow them to be applied outside the areas designated $(App_{\tau,\sigma})_{\tau,\sigma \in \mathcal{T}}$ and $(App_\tau)_{\tau \in \mathcal{T}}$ too, but this does not affect the satisfaction relation.

A *3-Henkin model* H is a a tuple $(\mathcal{T}, \mathcal{D}, \to, App, TApp, \mathcal{I}_{type}, \mathcal{I}, typeOf)$ together with a pair $((H_T)_{T \in TTerm}, (H_{tj})_{tj \in Tj, \vdash tj})$, where:
(a) $\to$: $\mathcal{T} \times \mathcal{T} \to \mathcal{T}$ is an injective mapping,
(b) $App : \mathcal{D} \times \mathcal{D} \to \mathcal{D}$,
(c) $TApp : \mathcal{D} \times \mathcal{T} \to \mathcal{D}$,
(d) $\mathcal{I}_{type} : TConst \to \mathcal{T}$,
(e) $typeOf \subseteq \mathcal{D} \times \mathcal{T}$,
(f) $H_T : \mathcal{T}^{\overline{TVar}} \to \mathcal{T}$ for each $T \in TTerm$,
(g) $H_{\Gamma \rhd X:T} : \{(\gamma, \delta) \in \mathcal{T}^{TVar} \times \mathcal{D}^{DVar} : \text{for all } x : T' \in \Gamma,\ typeOf(\delta(x),\ H_{T'}(\gamma))\} \to \mathcal{D}$ for each $\Gamma \rhd X : T \in Tj$ with $\vdash_{SF} \Gamma \rhd X : T$,
such that the following hold:
(1) $\{d \in \mathcal{D} : typeOf(d, \tau)\} \cap \{d \in \mathcal{D} : typeOf(d, \sigma)\} = \emptyset$ whenever $\tau \neq \sigma$;
(2) For each τ, σ, d, d', if $typeOf(d, \tau \to \sigma)$ and $typeOf(d', \tau)$ then $typeOf(App(d, d'), \sigma)$;
(3) For each t, T, γ, τ, d, if $typeOf(d, H_{\Pi t.T}(\gamma))$ then $typeOf(TApp(d, \tau),\ H_T(\gamma[t \leftarrow \tau]))$;
(4) For each τ, σ, *App* is (τ, σ)-extensional, i.e., for each d, d' with $typeOf(d, \tau \to \sigma)$ and $typeOf(d', \tau \to \sigma)$, if $App(d, d'') = App(d', d'')$ for all d'' with $typeOf(d'', \tau)$ then $d = d'$;
(5) For each T, t, γ, *TApp* is $H_{\Pi t.T}(\gamma)$-extensional, i.e., for each d, d' with $typeOf(d, \tau)$ and $typeOf(d', \tau)$, if $TApp(d, \sigma) = TApp(d', \sigma)$ for all σ, then $d = d'$;
(6) $H_t(\gamma) = \gamma(t)$ for each $t \in TVar$;
(7) $H_{tc}(\gamma) = \mathcal{I}_{type}(tc)$ for each $tc \in TConst$;
(8) $H_{T \to T'}(\gamma) = H_T(\gamma) \to H_{T'}(\gamma)$;
(9) $H_{\Gamma \rhd x:T}(\gamma, \delta) = \delta(x)$
(10) $H_{\Gamma \rhd XY:T}(\gamma, \delta) = App(H_{\Gamma \rhd X:T' \to T}(\gamma, \delta),\ H_{\Gamma \rhd Y:T'}(\gamma, \delta))$;
(11) $H_{\Gamma \rhd XT:T'[T/t]}(\gamma, \delta) = TApp(H_{\Gamma \rhd X:\Pi t.T'}(\gamma, \delta),\ H_T(\gamma))$;
(12) $typeOf(H_{\Gamma \rhd \lambda x:T.X:T \to T'}(\gamma, \delta),\ H_T(\gamma) \to H_{T'}(\gamma))$ and, for each d with $typeOf(d,\ H_T(\gamma))$, $App(H_{\Gamma \rhd \lambda x:T.X:T \to T'}(\gamma, \delta),\ d) = H_{\Gamma \cup \{x:T\} \rhd X:T'}(\gamma, \delta[x \leftarrow d])$;
(13) $typeOf(H_{\Gamma \rhd \lambda t.X:\Pi t.T}(\gamma, \delta), H_{\Pi t.T}(\gamma))$ and, for each $\tau \in \mathcal{T}$, $App(H_{\Gamma \rhd \lambda t.X:(\Pi t)T}(\gamma, \delta),\ \tau) = H_{\Gamma \rhd X:T}(\gamma[t \leftarrow \tau], \delta)$.

LEMMA 7. *2-Henkin and 3-Henkin models are equivalent, in that there exist two satisfaction- preserving and reflecting mappings $\&_3^2$ and $\&_2^3$ between the two classes of models.*

We are now ready to eliminate typing judgements from the semantics. The following lemma shows that typing judgements are semantically redundant:

LEMMA 8. *Let H be a 3-Henkin model, $\gamma : TVar \to \mathcal{T}$ and $\delta : DVar \to \mathcal{D}$. Then there for any two pairs (Γ, T) and (Γ', T') such that $\vdash \Gamma \rhd X : T$ and $\vdash \Gamma' \rhd X : T'$ such that $H_{\Gamma \rhd X:T}$ and $H_{\Gamma' \rhd X:T'}$ are defined on (γ, δ), it holds that $H_{\Gamma \rhd X:T}(\gamma, \delta) = H_{\Gamma' \rhd X:T'}(\gamma, \delta)$.*

Based on this lemma and on the fact that satisfaction is not affected by allowing interpretations of data terms that cannot type, we obtain some further equivalent models:

A *4-Henkin model* H is a a tuple $(\mathcal{T}, \mathcal{D}, \to, App, TApp, \mathcal{I}_{type}, \mathcal{I}, typeOf)$ together with a pair $((H_T)_{T \in TTerm}, (H_X)_{X \in DTerm})$, where:
(a) $\to$: $\mathcal{T} \times \mathcal{T} \to \mathcal{T}$ is an injective mapping,
(b) $App : \mathcal{D} \times \mathcal{D} \to \mathcal{D}$,
(c) $TApp : \mathcal{D} \times \mathcal{T} \to \mathcal{D}$,
(d) $\mathcal{I}_{type} : TConst \to \mathcal{T}$,
(e) $typeOf \subseteq \mathcal{D} \times \mathcal{T}$,
(f) $H_T : \mathcal{T}^{\overline{TVar}} \to \mathcal{T}$ for each $T \in TTerm$,
(g) $H_X : \mathcal{T}^{TVar} \times \mathcal{D}^{DVar} \to \mathcal{D}$ for each $X \in DTerm$,
such that the following hold:
(1) $\{d \in \mathcal{D} : typeOf(d, \tau)\} \cap \{d \in \mathcal{D} : typeOf(d, \sigma)\} = \emptyset$ whenever $\tau \neq \sigma$;
(2) For each τ, σ, d, d', if $typeOf(d, \tau \to \sigma)$ and $typeOf(d', \tau)$, then $typeOf(App(d, d'), \sigma)$;
(3) For each t, T, γ, τ and d with $typeOf(d, H_{\Pi t.T}(\gamma))$, it holds that $typeOf(TApp(d, \tau),\ H_T(\gamma[t \leftarrow \tau]))$;
(4) For each τ, σ, *App* is (τ, σ)-extensional, i.e., for each d, d' with $typeOf(d, \tau \to \sigma)$ and $typeOf(d', \tau \to \sigma)$, if $App(d, d'') = App(d', d'')$ for all d'' with $typeOf(d'', \tau)$ then $d = d'$;
(5) For each T, t, γ, *TApp* is $H_{\Pi t.T}(\gamma)$-extensional, i.e., for each d, d' with $typeOf(d, H_{\Pi t.T}(\gamma))$ and $typeOf(d', \tau)$, if $TApp(d, \sigma) = TApp(d', \sigma)$ for all σ, then $d = d'$;
(6) $H_t(\gamma) = \gamma(t)$ for each $t \in TVar$;
(7) $H_{tc}(\gamma) = \mathcal{I}_{type}(tc)$ for each $tc \in TConst$;
(8) $H_{T \to T'}(\gamma) = H_T(\gamma) \to H_{T'}(\gamma)$;
(9) $H_x(\gamma, \delta) = \delta(x)$
(10) $H_{XY}(\gamma, \delta) = App(H_X(\gamma, \delta),\ H_Y(\gamma, \delta))$;
(11) $H_{XT}(\gamma, \delta) = TApp(H_{\Pi t.T'}(\gamma, \delta),\ H_T(\gamma, \delta))$;
(12) If $typeOf(H_{\lambda x:T.X}(\gamma, \delta),\ H_T(\gamma) \to H_{T'}(\gamma))$ and $typeOf(d,\ H_T(\gamma))$, then $App(H_{\lambda x:T.X}(\gamma, \delta),\ d) = H_X(\gamma, \delta[x \leftarrow d])$;
(13) if $typeOf(H_{\lambda t.X}(\gamma, \delta), H_{\Pi t.T}(\gamma))$ then $App(H_{\lambda t.X}(\gamma, \delta),\ \tau) = H_X(\gamma[t \leftarrow \tau], \delta)$.

In the definition of 4-Henkin models, H_T has $\mathcal{T}^{TVar}$ as its domain. If we were to regard such models as two-sorted GFOL models, we would need its domain to be $\mathcal{T}^{TVar} \times \mathcal{D}^{DVar}$, just like the one of H_X. And indeed H_T can be seen as a function on the latter domain, constant in the second variable; this view of H_T is consistent with the properties of GFOL models, since H_T should depend solely on the free variables of T, and there are no data variables occurring in a type term. Thus 4-Henkin models are immediately equivalent to the following:

A *5-Henkin model* H is a a tuple $(\mathcal{T}, \mathcal{D}, \to, App, TApp, \mathcal{I}_{type}, \mathcal{I}, typeOf)$ together with a pair $((H_T)_{T \in TTerm}, (H_X)_{X \in DTerm})$, where:
(a) $\to$: $\mathcal{T} \times \mathcal{T} \to \mathcal{T}$ is an injective mapping,
(b) $App : \mathcal{D} \times \mathcal{D} \to \mathcal{D}$,

(c) $\mathit{TApp} : \mathcal{D} \times \mathcal{T} \to \mathcal{D}$,
(d) $\mathcal{I}_{type} : \mathit{TConst} \to \mathcal{T}$,
(e) $\mathit{typeOf} \subseteq \mathcal{D} \times \mathcal{T}$,
(f) $H_T : \mathcal{T}^{\mathit{TVar}} \times \mathcal{D}^{\mathit{DVar}} \to \mathcal{T}$ for each $T \in \mathit{TTerm}$,
(g) $H_X : \mathcal{T}^{\mathit{TVar}} \times \mathcal{D}^{\mathit{DVar}} \to \mathcal{D}$ for each $X \in \mathit{DTerm}$,
such that the following hold:
(1) $\{d \in \mathcal{D} : \mathit{typeOf}(d, \tau)\} \cap \{d \in \mathcal{D} : \mathit{typeOf}(d, \sigma)\} = \emptyset$ whenever $\tau \neq \sigma$;
(2) For each τ, σ, d, d', if $\mathit{typeOf}(d, \tau \to \sigma)$ and $\mathit{typeOf}(d', \tau)$, then $\mathit{typeOf}(\mathit{App}(d, d'), \sigma)$;
(3) For each $t, T, \gamma, \tau, \delta$ and d with $\mathit{typeOf}(d, H_{\Pi t.T}(\gamma, \delta))$, it holds that $\mathit{typeOf}(\mathit{TApp}(d, \tau), H_T(\gamma[t \leftarrow \tau], \delta))$;
(4) For each τ, σ, App is (τ, σ)-extensional, i.e., for each d, d' with $\mathit{typeOf}(d, \tau \to \sigma)$ and $\mathit{typeOf}(d', \tau \to \sigma)$, if $\mathit{App}(d, d'') = \mathit{App}(d', d'')$ for all d'' with $\mathit{typeOf}(d'', \tau)$ then $d = d'$;
(5) For each T, t, γ, δ, TApp is $H_{\Pi t.T}(\gamma, \delta)$-extensional, i.e., for each d, d' with $\mathit{typeOf}(d, H_{\Pi t.T}(\gamma, \delta))$ and $\mathit{typeOf}(d', \tau)$, if $\mathit{TApp}(d, \sigma) = \mathit{TApp}(d', \sigma)$ for all σ, then $d = d'$;
(6) $H_t(\gamma, \delta) = \gamma(t)$ for each $t \in \mathit{TVar}$;
(7) $H_{tc}(\gamma, \delta) = \mathcal{I}_{type}(tc)$ for each $tc \in \mathit{TConst}$;
(8) $H_{T \to T'}(\gamma, \delta) = H_T(\gamma, \delta) \to H_{T'}(\gamma, \delta)$;
(9) $H_x(\gamma, \delta) = \delta(x)$
(10) $H_{XY}(\gamma, \delta) = \mathit{App}(H_X(\gamma, \delta),\ H_Y(\gamma, \delta))$;
(11) $H_{XT}(\gamma, \delta) = \mathit{TApp}(H_{\Pi t.T'}(\gamma, \delta),\ H_T(\gamma, \delta))$;
(12) If $\mathit{typeOf}(H_{\lambda x:T.X}(\gamma, \delta),\ H_T(\gamma, \delta) \to H_{T'}(\gamma, \delta))$ and $\mathit{typeOf}(d,\ H_T(\gamma, \delta))$, then $\mathit{App}(H_{\lambda x:T.X}(\gamma, \delta),\ d) = H_X(\gamma, \delta[x \leftarrow d])$;
(13) if $\mathit{typeOf}(H_{\lambda t.X}(\gamma, \delta), H_{\Pi t.T}(\gamma, \delta))$ then $\mathit{App}(H_{\lambda t.X}(\gamma, \delta),\ \tau) = H_X(\gamma[t \leftarrow \tau], \delta)$.

It should be clear that 5-Henkin models are essentially two-sorted GFOL models satisfying $\mathcal{SF}$, modulo a discussion similar to the one we had on untyped λ-calculus. Note also that now we can eliminate the disjointness assumption (1), as well as the injectivity assumption about $\to$, since these would not affect the satisfaction of GFOL Horn clauses with conclusion referring to data, the only ones that we are interested in. We have thus obtained that Henkin models of System F are equivalent to the GFOL models of $\mathcal{SF}$.

Henkin Models versus GFOL Models, Compactly. Above we showed the relationship between the Henkin and GFOL semantics for System F by describing a multi-step process that slowly made us view a model of the former as a model of the later. While such a presentation has the advantage that it provides a convincing argument for the equivalence of the two semantics (since each of its steps was relatively simple), it might nevertheless lose sight of the resulting correspondence between models. Next we provide the direct version of the correspondence, together with the lemmas asserting its correctness and together with a theorem stating rigorously that the two semantics are equivalent (we state these results without any proofs, as they just describe compactly the situation presented and justified above).

Henkin to GFOL: For each Henkin model H, we define a GFOL model $M = H^{\#}$ as follows:
(a) $M_{type} = \mathcal{T}$;
(b) $M_{data} = \bigcup \mathcal{D}om$;
(c) for each $T \in \mathit{TTerm}$, $M_T : \mathit{Map}(\mathit{Var}, M) \to M_{type}$ is the mapping given by $M_T(\gamma, \delta) = H_T(\gamma)$ (recall that $\mathit{Map}(\mathit{Var}, M)$ denotes the set of two-sorted functions from $\mathit{Var} = (\mathit{TVar}, \mathit{DVar})$ to $M = (M_{type}, M_{data})$);
(d) for each $X \in \mathit{DExp}$, $M_X : \mathit{Map}(\mathit{Var}, M) \to M_{data}$ is the mapping given by: $M_X(\gamma, \delta) = H_{\Gamma \triangleright X:T}(\gamma, \delta)$ if there exists Γ and T such that $\vdash_{SF} \Gamma \triangleright X : T$ and $H_{\Gamma \triangleright X:T}$ is defined on (γ, δ); otherwise we take an arbitrary value in $\bigcup \mathcal{D}om$;
(e) $(d, \tau) \in B_{\mathit{typeOf}}$ iff $d \in \mathcal{D}om_\tau$.

Roughly, $H^{\#}$ is obtained from H by throwing types on the sort *type* and all data of any type on sort *data*; the relation *typeOf* keeps the connection between well-typed terms and types as in H. $H^{\#}$ may also contain some additional "junk", of typeless (error) data.

LEMMA 9. *The mappings M_X above are well-defined for all $X \in \mathit{DExp}$. More specifically, for each γ, δ, $H_{\Gamma \triangleright X:T}(\gamma, \delta)$ does not depend on the choice of Γ and T, so long as $\vdash_{SF} \Gamma \triangleright X : T$ and $H_{\Gamma \triangleright X:T}$ is defined on (γ, δ). Moreover, M is indeed a GFOL model and $M \models_{GFOL} \mathcal{SF}$.*

GFOL to Henkin: For each GFOL model M satisfying $\mathcal{SF}$, we define a Henkin model $H = M^{\$}$ as follows:
(a) $\mathcal{T} = M_{type}$;
(b) $\mathcal{F} = \{\tau \mapsto M_T(\gamma[t \leftarrow \tau], \delta) : \gamma, \delta, T \text{ arbitrary}\}$;
(c) $\tau \to \sigma = M_{t \to t'}(\gamma, \delta)$, where $\gamma(t) = \tau$, $\gamma(t') = \sigma$;
(d) $\forall(f) = M_{(\forall t)T}(\gamma, \delta)$, if f is $\tau \mapsto M_T(\gamma[t \leftarrow \tau], \delta)$;
(e) $\mathcal{I}_{type}(tc) = M_{tc}(\gamma, \delta)$ for some arbitrary (γ, δ);
(f) $\mathcal{D}om_\tau = \{d \in M_{data} : (d, \tau) \in M_{\mathit{typeOf}}\}$;
(g) $\mathit{App}_{\tau,\sigma}(d)(d') = M_{x\,y}(\gamma, \delta)$, where $\delta(x) = d$, $\delta(y) = d'$;
(h) $\mathit{App}_f(d)(\tau) = M_{x\,t}(\gamma, \delta)$ where $\gamma(t) = \tau$ and $\delta(x) = d$;
(i) $\mathcal{I}(dc) = M_{dc}(\gamma, \delta)$ for some arbitrary (γ, δ);
(j) $H_T(\gamma) = M_T(\gamma, \delta)$ for some arbitrary δ;
(k) $H_{\Gamma \triangleright X:T}(\gamma, \delta) = M_X(\gamma, \delta)$. Roughly, $M^{\$}$ classifies the well-typed data, on types, according to the *typeOf* relation in M.

LEMMA 10. *All the above mappings are well-defined, i.e.:*
- $M_{t \to t'}(\gamma, \delta)$ does not depend on the choice of γ, δ, so long as $\gamma(t) = \tau$, $\gamma(t') = \sigma$;
- if for some $T, T', t, t', \gamma, \gamma', \delta, \delta'$, the mappings $\tau \mapsto M_T(\gamma[t \leftarrow \tau], \delta)$ and $\tau \mapsto M_{T'}(\gamma[t \leftarrow \sigma], \delta)$ coincide, then $M_{(\forall t)T}(\gamma, \delta) = M_{(\forall t')T'}(\gamma', \delta')$;
- $M_{tc}(\gamma, \delta)$ and $M_{tc}(\gamma, \delta)$ do not depend on the choice of γ, δ;
- $M_{x\,y}(\gamma, \delta)$ does not depend on the choice of γ, δ, so long as $\delta(x) = d$, $\delta(y) = d'$; and if $(d, \tau \to \sigma), (d', \tau) \in M_{\mathit{typeOf}}$, then $(M_{x\,y}(\gamma, \delta), \sigma) \in M_{\mathit{typeOf}}$;
- $M_T(\gamma, \delta)$ does not depend on the choice of δ.
Moreover, H defined above is indeed a Henkin model.

Note that the mappings $\$$ and $\#$ are actually the compositions of all the intermediate mappings between the different versions of Henkin models discussed above.

PROPOSITION 9. *Assume $\vdash_{SF} \Gamma \triangleright X : T$ and $\vdash_{SF} \Gamma \triangleright Y : T$. Then:*

(1) $H \models_{SF} \Gamma \triangleright X = Y : T$ iff $H^{\#} \models_{GFOL} \Gamma^{\#} \Rightarrow X = Y$;
(2) $M \models_{GFOL} \Gamma^{\#} \Rightarrow X = Y$ iff $M^{\$} \models_{SF} \Gamma \triangleright X = Y : T$.

Summing up the situation described by Propositions 8 and 9:

1. The well-typed System-F terms are precisely the ones that denote typed data (i.e., data items d such that there exists a type τ with $\mathit{typeOf}(d, \tau)$) in all models of $\mathcal{SF}$;
2. Henkin models correspond to models of $\mathcal{SF}$, and vice versa;
3. A well-typed System-F equation is represented by a conditional equation in GFOL;
4. A well-typed equation is satisfied by a Henkin model iff it is satisfied by its corresponding model of $\mathcal{SF}$, and vice versa.

5. Related Work

To properly distinguish GFOL from other approaches, a discussion on *encodings* of formal systems is appropriate. Consider a formal system, consisting of a syntax and deduction rules, say λ-calculus, denoted Λ. One can formally mimic the informal definition of Λ *inside* an axiomatic set theory, say ZF, by setting ZF-formulae

$var(x)$, $term(x)$ and $eq(x,y)$, which say "x is a (λ-calculus) variable", "x is a term", and "x, y are terms and their equality is deducible in Λ", etc. Then any proof *about* Λ, including proofs that certain facts are deducible *in* Λ, can be carried on inside ZF. However, as far as the original system Λ is concerned, this ZF "definition" of Λ is *not a definition, but an encoding*; and to make the encoding rigorous, one should give a mapping from terms in Λ to their ZF representations.

Assume that the language of ZF has included, via a process of Skolemization, the constants 0, *succ*, *app*, and *lam*, together with axioms stating their desired meaning: "*succ* is the successor function on natural numbers", "*app* is the function that takes terms t_1, t_2 to $t_1 t_2$", etc.; also, assume that, both in Λ and in ZF, one uses natural numbers as λ-calculus variables. The encoding Enc : Λ-terms $\rightarrow$ ZF-terms can be defined by $Enc(n) = succ.....succ(0)$ (n times), $Enc(t_1 t_2) = app(Enc(t_1), Enc(t_2))$, $Enc(\lambda n.t) = lam(Enc(n), Enc(t))$. The encoding is faithful: for λ-terms t_1, t_2, $\vdash_\Lambda t_1 = t_2$ iff $ZF \vdash Enc(t_1) = Enc(t_2)$. Thus ZF can be used as a device to prove equalities in Λ. Yet, ZF does *not generalize* Λ, nor is Λ a ZF theory or anything similar. Λ was only encoded in ZF, so ZF regards it *as an object about which it can reason*.

What an encoding of a formal system cannot provide is a *meaningful* model-theoretic semantics for that system, because the encoding was only concerned with representing syntax and deduction. Indeed, given a presumptive model of ZF,[8] one cannot claim that it provides a model for the λ-calculus. A ZF model would merely provide a universe where the set of Λ-terms and the Λ deductive system would dwell. It is true that any faithful encoding translates whatever complete models the meta-theory (here ZF) has into models of the object-theory (here Λ); e.g., by the completeness of FOL where ZF is a theory, $\vdash_\Lambda t_1 = t_2$ iff $M \models Enc(t_1) = Enc(t_2)$ for all models M of ZF. But such a "semantics" would clearly be unacceptable for Λ, as a ZF model contains representations of λ-terms, deduction rules, proofs, and the whole syntactic infrastructure of Λ; moreover, this "semantics" is complete not because Λ equalities are stating facts *about* these models, as would be desirable with a semantics, but simply because the ZF models contain structure that mimics Λ deductions.

The point of this discussion is independent of Λ and ZF; it applies to any encoding. In particular, whenever one uses a *fixed* calculus or logic, such as HOL, to encode any other calculi, one cannot claim to provide models for them. Thus, e.g., the fact that HOL admits complete Henkin models and also can encode virtually any formal system, does not mean that its Henkin models work as a uniform semantics for these systems.

HORN^2 (and thus GFOL) relates to its defined calculi *not* via syntactic encodings, but via meaningful semantic interpretations. It does not regard the desired calculus as a formal system that needs to be somehow encoded; for instance, it does not make any attempt to encode the structure of typing environments as such, but rather regards them as syntactic counterparts of a "higher-level" semantic intuition, just like, e.g., any FOL deductive system with all its awkward side-conditions can be seen as a reflection of a meaningful FOL model theory. Some of the related works that we discuss next fall in the category of encodings.

Higher-Order Abstract Syntax (HOAS). In HOAS [19, 11], one uses a fixed λ-calculus, the *meta-calculus*, to encode various other formal systems, such as calculi or deductive systems of logics - let us refer to these as *object systems*. All the syntactic categories of an object system (terms, formulae, proofs, evaluation relations) become terms in the meta-calculus. Object-system bindings are represented by λ-bindings in the meta-calculus. In particular, quite different binding operators in the object system (such as λ-abstraction and universal quantification) are represented uniformly, using a single binding operator – the meta-level λ-abstraction. Consequently, all object-calculus axiom schemes become simple axioms in the meta-calculus, their "schematic aspect" being handled by a a built-in meta-calculus scheme, which does not appear in the specification itself. (To the contrary, GFOL specifications need to use axiom schemes in order to capture directly the axiom schemes of the defined calculus - see Appendix A.) HOAS' syntactically uniform representation, though very useful for proof-theoretic aspects, cannot be sensitive to the model-theoretic aspects of the object system. In particular, some presumptive models of the meta-calculus do not provide models for the object calculus; and indeed, HOAS does not attempt to provide such models, being concemed mainly with proof-theoretic adequacy. Take for instance a representation of λ-calculus in Edinburgh LF - such a representation would define the type of terms, that of equations, and the dependent types of proofs. Thus a presumptive model of this theory (consisting here only of constant declarations) stated in the dependent-type calculus of LF, would be dwelled by elements called terms, equations, and proofs, hence it would be far from being an appropriate model of λ-calculus; the latter should surely not provide any interpretations for proofs as elements in the model (though, as argued in [2], an LF representation, if chosen in a "denotational" manner, could *suggest* a model).

In conclusion, unlike GFOL, HOAS falls into the encoding-based approaches, even though the encoded object has an extra affinity with its encoding, as they share the same bindings. On the more operational side, one can define any meta-calculi used in HOAS (such as untyped λ-calculus and the Edinburgh LF λ-calculus with dependent types) as HORN^2 theories and then use these theories to represent object systems the same way the original calculi do, but this would *not* be a proper use of HORN^2 - as mentioned, its technique for defining calculi is different from HOAS. Also, a framework such as LF could easily encode any instance of GFOL and its theories (pretty much like it encodes FOL), whenever the underlying term syntax is encodable.

Nominal Logic (NL) [21] is a first-order logic that deals with abstract syntax by means of *names*, which can be bound in terms just like λ-calculus variables, but they are *not* variables; names are *semantical entities*, having syntax-independent meaning and being addressed explicitly by the freshness relation and the swapping operator. GFOL resembles NL in that both are parameterized by a notion of a term (substitution-based for GFOL, binding-based for NL) and both are first-order. However, GFOL differs from NL in several aspects: **(1)** The NL approach to calculi definitions is even more encoding-based than HOAS, as it explicitly defines substitution and freshness *inside its theories*. **(2)** The NL models contain semantic support for substitution, in its more amenable form of swapping/permutation of names; GFOL models are required to interpret syntax in a way that is substitution-consistent, but not to interpret substitution as such. **(3)** NL is *not* a complete logic, due to the second-order nature of the restrictions on its models.

Explicitly Closed Families and Binding Algebras. Structures consisting of *explicitly closed families* and *functionals* (ECFF) were introduced in [1] and studied as *binding algebras* in [25]. An ECFF consists of a set A together with a family of operations on A and a family of functionals (mappings between functions), such that the set of functions on A is closed under the functional-based polynomial combinators. The notion of term in this frameworks forms a term syntax in our sense, and ECFFs are particular cases of models

[8] The fact that ZF cannot be proven to have models is irrelevant for this discussion - we could have chosen weaker systems instead.

in our sense. However, ECFF have a more restricted use than our GFOL models, since they make the commitment that bindings always define functions. To the contrary, the interpretation of terms in GFOL models has a loose character; it does not make any commitment other than what is prescribed by the axioms of the theories. As a consequence, calculi involving types of bindings other than "functional" find a direct representation in GFOL, while in ECFF they could only hope for a functional encoding.

Substitution Algebras [6] treat substitution abstractly like we do, but require models to account for substitution in a direct way. Therefore they need to work in a presheaf topos different from Set as the underlying universe, in order for elements in models to be *families* of items, sensitive to the change of context/environment. (Thus they propose a solution for "semantic substitution" different than Nominal Logic.) While *substitution algebras* are related to our term syntaxes, models differ in that we do *not* require them to have built-in "abstract syntax" on their carrier sets, but rather to be able to provide interpretations for all syntactic features. As opposed to substitution algebras, HORN^2 is directly applicable to typed calculi, with typing judgements that change the typing context being captured by HORN^2 formulae.

6. Concluding remarks

We defined a generic first-order logic, GFOL, in which terms are axiomatized by common properties of their free variables and substitution, together with a complete deduction system. A fragment of GFOL with sentences more general than Horn, called HORN^2, was shown to admit a more effective complete Gentzen system. Several λ-calculi were defined as theories in HORN^2, following a "higher-level" view that allows one to focus on the specific aspects of the calculus rather than on syntactic or tautological details. This higher-level view brings a complete semantics to the specified calculi in a natural, meaningful, and uniform way.

The kind of semantics that a calculus receives via GFOL is usually called *loose*, or *logical* semantics. This means that a calculus does not receive a *denotation* in a fixed model, but is rather regarded as stating axioms about a whole class of models, just like group theory states axioms that are to hold in a class of models called groups. Completeness of a loose semantics means that a statement in the language of the calculus is derivable in the calculus iff it is true in all models. Thus one might be tempted to say that loose models capture faithfully what the calculus can prove. However, a loose semantics, while convenient for many purposes, is by no means the end of the "semantic story" of the calculus, as sometimes a calculus hides inside more than its deductive system can prove - hence the need for a denotation that would provide further insight into what the calculus actually "means", in particular would discover desired properties that were implicit in the calculus in a way more subtle than by bare deduction consequence.

Regarding a presumptive denotational-semantics methodology developed on top of HORN^2, this was not the concern of the present paper, but seems like a promising subject for future research. The key to this would be the study of appropriate free and initial models for certain HORN^2 theories - these models would constitute the desired denotation in a similar style with initial models constituting the "desired denotation" for a first-order data-type specification. For example, extensional theories can be shown to admit free models, as well as initial reachable models, along the lines of the interesting results from [16]. To faithfully capture higher-order denotations, partiality might be necessary as a first-class citizen in the models, via incomplete interpretations of terms into models.

Finally, it would be challenging to also study computational, i.e., operational, aspects that could be extracted from the HORN^2 theories: can equations and some relations such as *typeOf* or *moreGeneral* be "executed", e.g., via rewriting or some form of general-purpose logic programming technique, and thus obtain a calculus-independent operational semantics methodology?

References

[1] P. Aczel. Frege structures and notations in propositions, truth and set. In *The Kleene Symposium*, pages 31–59. North Holland, 1980.

[2] A. Avron, F. Honsell, I. A. Mason, and R. Pollack. Using typed lambda calculus to implement formal systems on a machine. *J. of Automated Reasoning*, 9(3):309–354, 1992.

[3] H. P. Barendregt. *The Lambda Calculus*. North-Holland, 1984.

[4] G. Birkhoff. On the structure of abstract algebras. *Proceedings of the Cambridge Philosophical Society*, 31:433–454, 1935.

[5] K. B. Bruce, A. R. Meyer, and J. C. Mitchell. The semantics of second-order lambda calculus. *Information and Computation*, 85(1):76–134, 1990.

[6] M. Fiore, G. Plotkin, and D. Turi. Abstract syntax and variable binding (extended abstract). In *Proc. 14$^{\text{th}}$ LICS Conf.*, pages 193–202. IEEE, Computer Society Press, 1999.

[7] J. H. Gallier. *Logic for computer science. Foundations of automatic theorem proving*. Harper & Row, 1986.

[8] J.-Y. Girard. Une extension de l'interpretation de Gödel a l'analyse, et son application a l'elimination des coupure dans l'analyse et la theorie des types. pages 63–92. North Holland, 1971.

[9] J. Goguen and J. Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105(2):217–273, 1992.

[10] C. A. Gunter. *Semantics of Programming Languages. Structures and Techniqes*. MIT Press, 1992.

[11] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. In *Proc. 2$^{\text{nd}}$ LICS Conf.*, pages 194–204. IEEE, Computer Society Press, 1987.

[12] A. Kennedy. *Programming Languages and Dimension*. PhD thesis, University of Cambridge, 1996.

[13] S. Mac Lane. *Categories for the Working Mathematician*. Springer, 1971.

[14] R. Milner. A theory of type polymorphism in programming. *J. of Computer and System Sciences*, 17(3):348–375, 1978.

[15] J. C. Mitchell. *Foundations for Programming Languages*. MIT Press, 1996.

[16] B. Möller, A. Tarlecki, and M. Wirsing. Algebraic specifications of reachable higher-order algebras. In *ADT*, pages 154–169, 1987.

[17] J. D. Monk. *Mathematical Logic*. Springer-Verlag, 1976.

[18] T. Mossakowski, J. Goguen, R. Diaconescu, and A. Tarlecki. What is a logic? In J.-Y. Beziau, editor, *Logica Universalis*, pages 113–133. Birkhauser, 2005.

[19] F. Pfenning and C. Elliot. Higher-order abstract syntax. In *PLDI '88*, pages 199–208. ACM Press, 1988.

[20] B. C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

[21] A. M. Pitts. Nominal logic: A first order theory of names and binding. In *TACS'01*, volume 2215 of *Lecture Notes in Computer Science*, pages 219–242, 2001.

[22] A. Poigné. Another look at parameterization using algebras with subsorts. In *MFCS'84*, volume 176 of *Lecture Notes in Computer Science*, pages 471–479, 1984.

[23] J. C. Reynolds. Towards a theory of type structure. In *Programming Symposium, Proceedings Colloque sur la Programmation*, volume 19 of *Lecture Notes in Computer Science*, pages 408–423. Springer-Verlag, 1974.

[24] G. Roşu. Extensional theories and rewriting. In *ICALP'04*, volume

3142 of *Lecture Notes in Computer Science*, pages 1066–1079, 2004.

[25] Y. Sun. An algebraic generalization of Frege structures - binding algebras. *Theoretical Computer Science*, 211(1-2):189–232, 1999.

A. The Use of Axiom Schemes

Due to the presence of binding, stating properties such as the β-reduction amounts to giving an infinite number of axioms that follow a certain pattern, i.e., to providing an *axiom scheme*. This situation, present in all lambda calculi, persists in HORN^2 theories as well. While the use of axiom schemes (and therefore of infinite recursive axiomatizations rather than finite ones) is indeed a drawback, we consider that this is the necessary side-effect of the direct way in which HORN^2 approaches the definitions of calculi - not by *encoding* as in HOAS, but by *instanciating* directly to the desired calculus. In this sense, HORN^2 is not a "logical framework" for calculi definitions (as Edinburgh LF is), but rather a *generalization* for all these calculi (according to Section 3). Moreover, axiom schemes, so long as they rely on side-condition-free term patterns like all the axiom schemes of HORN^2 specifications tend to do, are not really a burden for deduction, as we argue below.

Indeed, the HORN^2- Gentzen systems discussed in Section 2.6 use axiom instances obtained from the axioms by substituting variables for terms. Now consider a term-pattern axiom scheme such as (T-β) for the System-F specification $\mathcal{SF}$ in Section 3: $\mathit{typeOf}(\lambda t.X, t') \Rightarrow (\lambda t.X)t = X$, where t, t' denote type variables and X an arbitrary data term (recall that the axiom is implicitly universally quantified over all its free variables). For using it inside the HORN^2 Gentzen system, one proceeds as follows: first one picks a term X, obtaining an instance of the axiom scheme; then one substitutes by terms all the free variables in this instance; finally, one places the substituted instance in an appropriate context in the sequents appearing in deduction rules. Note that, according to the above recipe, one needs to instanciate the axiom scheme twice: first by *replacing* the term metavariable X with an actual term, and then by *substituting* the resulted free variables with terms. However, following the usual practice of λ-calculi that do not discriminate term metavariables over variable metavariables, we can collapse these two instanciations into one, building on the intuitively clear (and easily checkable) fact that *after replacing a term meta-variable such as X with an actual term, we do not need to further substitute the free variables of this term, as we can take the already substituted term directly*. For example, in the HORN^2 Gentzen system $\mathcal{K}^{=}_{\mathcal{SF}}$ for amenable theories, (T-β) yields the following deduction rule, where T, T' stand for an arbitrary type terms:

$$\left| \frac{\Gamma \rhd \mathit{typeOf}(\lambda t.X, T')}{\Gamma \rhd (\lambda t.X)T = X[T/t]} \right| \text{(Inst-T-}\beta\text{)}$$

t above is substituted by T in the right of the succedent of the lower sequent only, since all the other occurrences of t are term-bound. We obtained the familiar (β)-rule for types. When transforming the axioms into their associated "instance" rules, side conditions may arise, as shown by the instance of $\mathcal{SF}$'s [Abs]:

$$\left| \frac{\Gamma\ \mathit{typeOf}(x, T) \rhd \mathit{typeOf}(X, T')}{\Gamma \rhd \mathit{typeOf}(\lambda x{:}T.X, T \to T')} \right| \text{[Inst-Abs]}$$

where x should be a data variable fresh w.r.t. Γ. Again we obtain the familiar typing rule for abstraction, with the familiar side-condition that x not be in Γ.

B. The Typing Relations

In the specifications of Section 3, typing was defined by means of a relation *typeOf* between the universe of data and the universe of types. We also spoke about *data terms* and *type terms*. This approach may look non-standard to the readers used to view types as items that are assigned to *terms*, at parsing or type-checking time, and not to data. We explain it next. Our specification methodology considers types, just like data, to be *semantic items*, populating the models. In this view, it is not the case that terms have types, but data items have types - and while *inferring* types for various data *in all models* one indeed uses terms (data terms and type terms), in a process that looks just like the traditional one of assigning types to terms. The one who wishes to regard typing purely syntactically can define the "syntactic typing relation" between terms (i.e., in our terminology, data terms) and types (i.e., type terms) as the following meta-relation: X has type T in the environment Γ iff the theory infers $\Gamma \rhd X : T$, i.e., if $\Gamma \rhd X : T$ is a tautological sequent w.r.t. the theory - this relation is indeed the expected least relation closed under some rules. But again, we regard typing judgements, just like the inferred equality between terms, loosely, as sentences that hold in all models. Just as much as we do not need (and makes no sense) to state in the theory that the equality relation is "the least one" closed under some rules, there is no need to make such meta-statements about typing either, since they hold by the very nature of deduction in any Gentzen system.

C. Sorts versus Types

HORN^2 is a many-sorted logic, i.e., has a many-sorted variant. When instanciating it to λ-calculi, depending on the complexity of the calculus and sometimes on mere taste, one may choose between two alternatives:

- To represent the types in the calculus as HORN^2 sorts. This way, typing is regarded *syntactically*, as a parsing, and thus meta-level, issue of HORN^2.
- To view types *semantically*, as inhabitants of some universe of types, and define the typing relation within the logic. This way, sorts are reserved for more general classifications of the semantic items, e.g. into *data* and *types*.

The second approach has the advantage of being more flexible, and thus covers the cases of more complex calculi with non-trivial typing and with higher-level classifiers such as kinds. We have pursued this approach in our specifications from Section 3. On the other hand, when possible, the first approach simplifies the structure of the formulae, since well-typed-ness need not be stated as an extra condition - indeed, extensional theories seem to suffice here, bringing, via Corollary 1, faithfulness of the representation for free, i.e., without the need to prove closure under the drop rules. We exemplify this first approach on an infinitely-sorted HORN^2 definition of simply-typed λ-calculus, alternative to $T\lambda$ of Section 3.

(Simply-)Typed λ-Calculus - Second Version ($T\lambda$II)

Let B be a set, of *basic types*.

- The sorts are $\mathit{Sort} ::= B \mid \mathit{Sort} \to \mathit{Sort}$ - let us call the sorts "types"
- Recall that for each $t, t' \in \mathit{Sort}$, $\mathit{Var}_t \cap \mathit{Var}_{t'} = \emptyset$
- For each $t, t' \in \mathit{Sort}$ and $b \in B$,

  $\mathit{Term}_b ::= \mathit{Var}_b \mid \mathit{Term}_{t \to b} \mathit{Term}_t$

  $\mathit{Term}_{t \to t'} ::= \mathit{Var}_{t \to t'} \mid \mathit{Term}_{t'' \to (t \to t')} \mathit{Term}_{t''} \mid \lambda \mathit{Var}_t.\mathit{Term}_{t'}$
- No relation symbol (except equality)

We let t, t', s range over types, x_t, y_t range over variables of type t, and X_t, Y_t range over terms of type t - note that here "term" means

"well-typed term". The theory is given below:

$(\forall x_t.X_s = Y_s) \Rightarrow \lambda x_t.X_s = \lambda x_t.Y_s$	(ξ)
$(\lambda x_t.X_s)x_t = X_s$	(β)
$x_{t\to t'} = \lambda y_t.x_{t\to t'}\ y_t$	(η)

The Gentzen system $\mathcal{K}^{=}_{T\lambda II}$ induced by this extensional theory is the following:

$\dfrac{\cdot}{X_t = X_t}$	(Inst-Refl)
$\dfrac{X_t = Y_t}{Y_t = X_t}$	(Inst-Symm)
$\dfrac{X_t = Y_t\,,\quad Y_t = Z_t}{X_t = Z_t}$	(Inst-Trans)
$\dfrac{X_t = Y_t}{Z_s[z_t \leftarrow X_t] = Z_s[z_t \leftarrow Y_t]}$	(Inst-Subst)
$\dfrac{X_s = Y_s}{\lambda x_t.X_s = \lambda x_t.Y_s}$	(Inst-ξ)
$\dfrac{\cdot}{(\lambda x_t.X_s)Y_t = X_s[x_t \leftarrow Y_t]}$	(Inst-β)
$\dfrac{\cdot}{X_{t\to s} = \lambda y_t.X_{t\to s}y_t}$	(Inst-η)

The above "low-level" implementation of $T\lambda II$ *is*, modulo a replacement of the congruence rules with a substitution rule, the typed λ-calculus itself. And its completeness holds immediately by Corollary 1, since $T\lambda II$ is an extensional theory. To the contrary, the previous Gentzen system for the theory $T\lambda$ needs a little work to be shown complete.

It is worth mentioning that the GFOL models of $T\lambda II$ are precisely the *Henkin models*, also called *frame models*, of simply-typed λ-calculus (see [15]).

D. Meta-Reasoning and Inductive Reasoning

Meta-reasoning about a calculus is not possible in its HORN^2 specification just as much as it is not possible in the calculus itself. Indeed, as already discussed, a HORN^2 *becomes* the specified calculus. In particular, one cannot show *in* HORN^2 that a calculus is confluent or terminating when equations are regarded as rewrite rules, neither that a programming language is deterministic. Even apart from their meta-theoretic aspect, these properties cannot be captured in an *axiomatic* approach like ours, where "evaluation" of a program to a value is only implicit in the deductive system, and not explicit as in an SOS or other forms of operational semantics.

Even if meta-reasoning is not available in the HORN^2 specifications (and is not meant to be), desired induction principles may be stated as sentences in an infinitary version of GFOL. For instance, here is how structural induction would look for the theory $U\lambda$ of Section 3 (that specifies untyped λ-calculus):

$((\bigwedge_{X\in Term} \forall FV(X)\setminus\{x\}.(\forall x.\varphi(X)) \Rightarrow \varphi(\lambda x.X)) \wedge$ $(\forall x, y.\varphi(x) \wedge \varphi(y) \Rightarrow \varphi(x\,y))) \Rightarrow \forall x.\varphi(x)$	(Ind)

Above, φ is an arbitrary GFOL formula with a pointed free variable. Note that this infinitary axiom is more manageable than it looks. It says that if one is able to prove $(\forall x.\varphi(X)) \Rightarrow \varphi(\lambda x.X)$ *for an arbitrary term* X and also to prove $\varphi(x)\wedge\varphi(y) \Rightarrow \varphi(x\,y))$, then one can infer $\forall x.\varphi(x)$.

Finally, desired existential properties, most notably existence of fixed points, are expressible and potentially provable in GFOL. For instance, a consequence in GFOL of $U\lambda$ is: $\forall x.\exists y.x\,y = y$, meaning that every item has, regarded as a function, a fixed point. Existence of "programs" performing desired tasks (i.e., conforming certain "specifications" stated as formulae) may also be expressed and proved directly in GFOL - a handy example is the existence of fixpoint operators: $\exists y.\forall x.x\,y\,x = y\,x$, a stronger version of the existence of fixed points.

E. Proofs

Here we give proofs or proof sketches for the results stated in the paper.

PROPOSITION 1. *The following hold:*

(1) $x \notin FV(T)$ *implies* $T[T'/x] = T$*;*
(2) $y[T/x] = T$ *if* $y = x$ *and* $y[T/x] = y$ *otherwise;*
(3) $FV(T[T'/x]) = FV(T)\setminus\{x\}\cup FV(T')$*;*
(4) $T[y/x][z/y] = T[z/x]$ *if* $y \notin FV(T)$*;*
(5) $T[y/x][x/y] = T$ *if* $y \notin FV(T)$*.*

Proof: We shall tacitly use properties (1)-(6) in Definition 1.
(1) Assume $x \notin FV(T)$. Since $[T'/x]\restriction_{FV(T)} = 1_{Var}\restriction_{FV(T)}$, we obtain $T[T'/x] = Subst(T, 1_{Var}) = T$.
(2) If $y = x$ then $y[T/x] = Subst(x, [T/x]) = [T/x](x) = T$. If $y \neq x$ then $[T/x]\restriction_{FV(y)} = 1_{Var}\restriction_{FV(y)}$, thus $y[T/x] = Subst(y, 1_{Var}) = y$.
(3) $FV(T[T'/x]) = FV(Subst(T, [T'/x])) =$
$\bigcup\{FV([T'/x](y)) : y \in FV(T)\} =$
$\bigcup\{FV([T'/x](x)) : x \in FV(T)\} \cup \bigcup\{FV([T'/x](y)) : y \in FV(T),\ y \neq x\} = FV(T') \cup (FV(T)\setminus\{x\})$.
Above, we also applied point (2) of the current proposition.
(4) We have that $T[y/x][z/y] = Subst(Subst(T, [y/x]), [z/y]) = Subst(T, [y/x]; [z/y])$. Now, for each $u \in Var$, we have that:

$$([y/x]; [z/y])(u) = Subst([y/x](u), [z/y]) =$$

$$= \begin{cases} Subst(y, [z/y]) & \text{, if } u = x \\ Subst(u, [z/y]) & \text{, if } u \neq x \end{cases} =$$

$$= \begin{cases} z, & \text{if } u = x \\ z, & \text{if } u \neq x \text{ and } u = y \\ u, & \text{if } u \neq x \text{ and } u \neq y \end{cases} =$$

$$= \begin{cases} z, & \text{if } u = x \text{ or } u = y \\ u, & \text{if } u \neq x \text{ and } u \neq y \end{cases}$$

Hence, since $y \notin FV(T)$, it follows that $[y/x][z/y]\restriction_{FV(T)} = [z/x]\restriction_{FV(T)}$, implying $Subst(T, [y/x]; [z/y]) = Subst(T, [z/x])$.
(5) It follows by point (4), since $Subst(T, [x/x]) = T$. □

PROPOSITION 2. *The following hold:*

(1) If $\rho\restriction_{FV(\varphi)} = \rho'\restriction_{FV(\varphi)}$*, then* $\rho \in A_\varphi$ *iff* $\rho' \in A_\varphi$*;*
(2) $\rho \in A_{Subst(\varphi,\theta)}$ *iff* $A_\theta(\rho) \in A_\varphi$*;*
(3) $\varphi \equiv_\alpha \psi$ *implies* $A_\varphi = A_\psi$*;*
(4) $\varphi \equiv_\alpha \psi$ *implies* $FV(\varphi) = FV(\psi)$*;*
(5) $\equiv_\alpha$ *is an equivalence;*
(6) $\varphi \equiv_\alpha Subst(\varphi, 1_{Var})$*;*
(7) $y \notin FV(\varphi)$ *implies* $\varphi[y/x][z/y] \equiv_\alpha \varphi[z/x]$*;*
(8) $x \notin FV(\varphi)$ *implies* $\varphi[T/x] \equiv_\alpha \varphi$*;*
(9) $\varphi \equiv_\alpha \psi$ *implies* $Subst(\varphi, \theta) \equiv_\alpha Subst(\psi, \theta)$*;*
(10) $\theta\restriction_{FV(\varphi)} = \theta'\restriction_{FV(\varphi)}$ *implies* $Subst(\varphi, \theta) \equiv_\alpha Subst(\varphi, \theta')$*;*
(11) $Subst(\varphi, \theta; \theta') \equiv_\alpha Subst(Subst(\varphi, \theta), \theta')$*;*
(12) $\varphi \equiv_\alpha \varphi'$ *and* $\psi \equiv_\alpha \psi'$ *implies:* $\neg\varphi \equiv_\alpha \neg\varphi'$, $\varphi\wedge\psi \equiv_\alpha \varphi'\wedge\psi'$, $\varphi \Rightarrow \psi \equiv_\alpha \varphi' \Rightarrow \psi'$, $\forall x.\varphi \equiv_\alpha \forall x.\varphi'$.

Proof: We shall tacitly use properties (1)-(6) in the definition of a term syntax and properties (c).(i-iii) in the definition of models. All proofs, except the one of point (12), will be performed by induction

either on the structure of formulae, or on the structure of $\equiv_\alpha$; "(IH)" will stand for the "Induction Hypothesis". Each time, we shall skip the case of logical connectors $\neg, \wedge, \Rightarrow$, since the induction step is trivial for them.

We prove (1) and (2) by induction on the structure of φ.

(1) Base case. $\rho\restriction_{FV(T_1,\ldots,T_n)} = \rho'\restriction_{FV(T_1,\ldots,T_n)}$ implies $\rho\restriction_{T_i} = \rho'\restriction_{T_i}$ for each $i \in \{1,\ldots,n\}$, which implies $A_{T_i}(\rho) = A_{T_i}(\rho')$ for each $i \in \{1,\ldots,n\}$, which implies that $\rho \in A_\varphi$ iff $\rho' \in A_\varphi$.

Induction step. Assume $\rho\restriction_{FV(\forall x.\varphi)} = \rho'\restriction_{FV(\forall x.\varphi)}$. Then $\rho\restriction_{FV(\varphi)\setminus\{x\}} = \rho'\restriction_{FV(\varphi)\setminus\{x\}}$, hence for all $a \in A$, $\rho[x \leftarrow a]\restriction_{FV(\varphi)} = \rho'[x \leftarrow a]\restriction_{FV(\varphi)}$. By (IH), we get that for all $a \in A$, $\rho[x \leftarrow a] \in A_\varphi$ iff $\rho'[x \leftarrow a] \in A_\varphi$, in particular that $\rho \in A_{\forall x.\varphi}$ iff $\rho' \in A_{\forall x.\varphi}$.

(2) Base case. We have the following equivalencies:
$\rho \in A_{Subst(\pi(T_1,\ldots,T_n),\theta)}$ iff
$\rho \in A_{\pi(Subst(T_1,\theta),\ldots,Subst(T_n,\theta))}$ iff
$(A_{Subst(T_1,\theta)}(\rho),\ldots,A_{Subst(T_n,\theta)}(\rho)) \in A_\pi$ iff
$(A_{T_1}(A_\theta(\rho)),\ldots,A_{T_n}(A_\theta(\rho))) \in A_\pi$ iff
$A_\theta(\rho) \in A_{\pi(T_1,\ldots,T_n)}$.

Induction step. We have the following equivalencies:

$\rho \in Subst(\forall x.\varphi,\theta)$ iff
$\rho \in A_{\forall z.Subst(\varphi,\theta[x\leftarrow z])}$ (where z is the least variable not in $FV(\varphi) \cup \bigcup\{\theta(y) : y \in FV(\varphi)\}$)[9] iff
$\rho[z \leftarrow a] \in A_{Subst(\varphi,\theta[x\leftarrow z])}$ for all $a \in A$, iff (by (IH))
$A_{\theta[x\leftarrow z]}(\rho[z \leftarrow a]) \in A_\varphi$ for all $a \in A$, iff (as will be proved shortly)
$A_\theta(\rho)[x \leftarrow a] \in A_\varphi$ for all $a \in A$, iff
$A_\theta(\rho) \in A_{\forall x.\varphi}$.

It remains to prove the promised equivalence. For it, it would suffice that $A_{\theta[x\leftarrow z]}(\rho[z \leftarrow a])\restriction_{FV(\varphi)} = A_\theta(\rho)[x \leftarrow a]\restriction_{FV(\varphi)}$. To prove the latter, let $y \in FV(\varphi)$. Then

$$A_{\theta[x\leftarrow z]}(\rho[x \leftarrow a])(y) = A_{\theta[x\leftarrow z](y)}(\rho[x \leftarrow a]) =$$

$$= \begin{cases} A_{\theta(y)}(\rho[z \leftarrow a]), & \text{if } x \neq y \\ A_z(\rho[z \leftarrow a]), & \text{if } x = y \end{cases} =$$

$$= \begin{cases} A_{\theta(y)}(\rho[z \leftarrow a]), & \text{if } x \neq y \\ a, & \text{if } x = y. \end{cases}$$

On the other hand,

$$A_\theta(\rho)[x \leftarrow a](y) = \begin{cases} A_{\theta(y)}(\rho), & \text{if } x \neq y \\ a, & \text{if } x = y. \end{cases}$$

Finally, we need to argue that $\rho[z \leftarrow a]\restriction_{FV(\theta(y))} = \rho\restriction_{FV(\theta(y))}$ - this is true because, by the choice of z, $z \notin FV(\theta(y))$.

Points (3)-(11) will be proved by induction on the structure of $\equiv_\alpha$.

(3): Base case. Obvious, since here $\equiv_\alpha$ coincides with equality.

Induction step. Assume $\forall x.\varphi \equiv_\alpha \forall y.\psi$. Then $\varphi[z/x] \equiv_\alpha \psi[z/y]$ for some $z \notin FV(\varphi) \cup FV(\psi)$. We have the following equivalencies:

$\rho \in A_{\forall x.\varphi}$ iff
$\rho[x \leftarrow a] \in A_\varphi$ for all $a \in A$, iff (as will be proved shortly)
$A_{[z/x]}(\rho[z \leftarrow a]) \in A_\varphi$ for all $a \in A$, iff (by point (2))
$\rho[z \leftarrow a] \in A_{\varphi[z/x]}$ for all $a \in A$, iff (by (IH))
$\rho[z \leftarrow a] \in A_{\psi[z/x]}$ for all $a \in A$, iff
$\rho \in A_{\forall x.\psi}$.

It remains to prove the promised equivalence. According to point (1), it would suffice that $\rho[x \leftarrow a]\restriction_{FV(\varphi)} = A_{[z/x]}(\rho[z \leftarrow a])\restriction_{FV(\varphi)}$.

To prove the latter, let $y \in FV(\varphi)$. Then

$$\rho[x \leftarrow a](y) =$$

$$\begin{cases} a, & \text{if } y = x \\ \rho(y), & \text{if } y \neq x. \end{cases}$$

On the other hand,

$$A_{[z/x]}(\rho[z \leftarrow a])(y) = A_{[z/x](y)}(\rho[z \leftarrow a]) =$$

$$= \begin{cases} A_z(\rho[z \leftarrow a]), & \text{if } y = x \\ A_y(\rho[z \leftarrow a]), & \text{if } y \neq x \end{cases} =$$

$$= \begin{cases} a, & \text{if } y = x \\ \rho[z \leftarrow a](y), & \text{if } y \neq x. \end{cases}$$

And since $z \notin FV(\varphi)$, $\rho(y) = \rho[z \leftarrow a](y)$.

(4): Base case. Obvious, since here $\equiv_\alpha$ coincides with equality.

Induction step. Assume $\forall x.\varphi \equiv_\alpha \forall y.\psi$, i.e., that $\varphi[z/x] \equiv_\alpha \psi[z/y]$ for some $z \notin FV(\varphi) \cup FV(\psi)$. By (IH), $FV(\varphi[z/x]) = FV(\psi[y/z])$, hence, by Proposition 1.(3), $FV(\varphi) \setminus \{x\} \cup \{z\} = FV(\psi) \setminus \{x\} \cup \{z\}$; because $z \notin FV(\varphi) \cup FV(\psi)$, this implies $FV(\varphi) \setminus \{x\} = FV(\psi) \setminus \{x\}$, i.e., $FV(\forall x.\varphi) = FV(\forall x.\psi)$.

Points (5)-(11) shall be proved *together*. In the case of (5), we prove by induction two properties - reflexivity and transitivity - since symmetry holds by definition.

Base case.

(5) Reflexivity and transitivity follow from the corresponding properties of equality.

(6) $Subst(\pi(T_1,\ldots,T_n),1_{Var}) = \pi(Subst(T_1,1_{Var}),\ldots,$ $Subst(T_n,1_{Var})) = \pi(T_1,\ldots,T_n)$, and thus
$Subst(\pi(T_1,\ldots,T_n,1_{Var}) \equiv_\alpha \pi(T_1,\ldots,T_n)$.

(7) and (8): Follow similarly to (6), but also using Proposition 1, points (4) and (1), respectively.

(9) Obvious, since here $\equiv_\alpha$ is the equality.

(10) and (11): Follow similarly to (6), using properties (3) and (9) in the definition of a term syntax.

Induction step:

(5) For reflexivity, note that $\forall x.\varphi \equiv_\alpha \forall x.\varphi$ holds because, by (IH) for point (5), $\varphi[z/x] \equiv_\alpha \varphi[z/x]$. In order to prove transitivity, assume that $\forall x_1.\varphi_1 \equiv_\alpha \forall x_2.\varphi_2$ and $\forall x_2.\varphi_2 \equiv_\alpha \forall x_3.\varphi_3$. Then for some $z \notin FV(\varphi_1) \cup FV(\varphi_2)$ and $z' \notin FV(\varphi_2) \cup FV(\varphi_3)$, it holds that $\varphi_1[z/x_1] \equiv_\alpha \varphi_2[z/x_2]$ and $\varphi_2[z'/x_2] \equiv_\alpha \varphi_3[z'/x_3]$. Let $z'' \notin FV(\varphi_1) \cup FV(\varphi_2) \cup FV(\varphi_3)$. Then, by (IH) for points (7) and (9), we have the following chain of α-equivalencies: $\varphi_1[z''/x_1] \equiv_\alpha \varphi_1[z/x_1][z''/z] \equiv_\alpha \varphi_2[z/x_2][z''/z] \equiv_\alpha \varphi_2[z''/x_1] \equiv_\alpha \ldots \equiv_\alpha \varphi_3[z''/x_3]$. From this, by (IH) for point (5), we get $\varphi_1[z''/x_1] \equiv_\alpha \varphi_3[z''/x_3]$; thus we found the desired $z'' \notin FV(\varphi_1) \cup FV(\varphi_3)$, yielding $\forall x_1.\varphi_1 \equiv_\alpha \forall x_3.\varphi_3$.

(6) We need to prove that $Subst(\forall x.\varphi,1_{Var}) \equiv_\alpha \forall x.\varphi$, i.e., that $\forall z.\varphi[z/x] \equiv_\alpha \forall x.\varphi$ with z as in the definition of substitution. Let $z' \notin FV(\varphi) \cup FV(\varphi[z/x])$, or equivalently, $z' \notin FV(\varphi) \cup \{z\}$. Then, by (IH) for point (7), $\varphi[z/x][z'/z] \equiv_\alpha \varphi[z'/x]$ and we are done.

(7) We need to prove that if $y \notin FV(\varphi)$, then $(\forall u.\varphi)[y/x][z/y] \equiv_\alpha (\forall u.\varphi)[z/x]$, i.e., $\forall u''.\varphi[u'/u][y/x][u''/u][z/y] \equiv_\alpha \forall u'''.\varphi[u'''/u][z/x]$, where u', u'', u''' are as in the definition of substitution for each of the three cases. Let $v \notin FV(\varphi) \cup \{u', u'', u'''\}$. It would suffice that $\varphi[u'/u][y/x][u''/u'][z/y][v/u''] \equiv_\alpha \varphi[u'''/u][z/x]v/u''']$; the latter is true by (IH) for point (11), since $[u'/u];[y/x];[u''/u'];[z/y];[v/u''] = [u'''/u];[z/x];[v/u''']$.

(8) We need to show that if $T \notin FV(\forall u.\varphi)$, then $(\forall u.\varphi)[T/x] \equiv_\alpha \forall u.\varphi$, i.e., that $\forall z.Subst(\varphi,[T/x][u \leftarrow z]) \equiv_\alpha \forall u.\varphi$, where z is as in the definition of substitution. Let $z' \notin FV(\varphi) \cup \{z\} \cup FV(T)$. It would suffice that $Subst(\varphi,[T/x][u \leftarrow z])[z'/z] \equiv_\alpha \varphi[z'/u]$, i.e., by (IH) for point (11), that $Subst(\varphi,[T/x][u \leftarrow z];[z'/z]) \equiv_\alpha$

[9] From now on, whenever we need to consider such a variable z, we just render it as "the variable from the definition of substitution".

$Subst(\varphi, [z'/u])$. The latter follows by (IH) for point (10), since $[T/x][u \leftarrow z']\restriction_{FV(\varphi)} = [z'/u]\restriction_{FV(\varphi)}$.
(9) Assume $\forall x.\varphi \equiv_\alpha \forall y.\psi$, i.e., that $\varphi[v/x] \equiv_\alpha \psi[v/y]$ for some $v \notin FV(\varphi) \cup FV(\psi)$. In order to prove $Subst(\forall x.\varphi, \theta) \equiv_\alpha Subst(\forall y.\psi, \theta)$, we take z, z' as in the definition of substitution and show that $\forall z.Subst(\varphi, \theta[x \leftarrow z]) \equiv_\alpha \forall z'.Subst(\psi, \theta[y \leftarrow z'])$. For proving the latter, using (IH) for point (11), we take $z'' \notin FV(\varphi) \cup FV(\psi)$ and show that $Subst(\varphi, \theta[x \leftarrow z]; [z''/z]) \equiv_\alpha Subst(\psi, \theta[y \leftarrow z']; [z''/z'])$, i.e., that $Subst(\varphi, \theta[x \leftarrow z'']) \equiv_\alpha Subst(\psi, \theta[y \leftarrow z''])$. Since $[z''/x]; \theta[x \leftarrow z''] = \theta[x \leftarrow z'']$ and $[z''/y]; \theta[y \leftarrow z''] = \theta[x \leftarrow z'']$, we reduce the desired equivalence to $Subst(\varphi, [z''/x]; \theta[x \leftarrow z'']) \equiv_\alpha Subst(\psi, [z''/y]; \theta[y \leftarrow z''])$, and furthermore, by (IH) for point (11), to $Subst(\varphi[z''/x], \theta[x \leftarrow z'']) \equiv_\alpha Subst(\psi[z''/y], \theta[y \leftarrow z''])$. Now, by (IH) for point (7), we have that $\varphi[z''/x] \equiv_\alpha \varphi[v/x][z''/v] \equiv_\alpha \psi[v/y][z''/v] \equiv_\alpha \psi[z''/y]$; moreover, by (IH) for point (10), from $\theta[x \leftarrow z'']\restriction_{FV(\varphi[z''/x])} = \theta\restriction_{FV(\varphi[z''/x])}$, we get $Subst(\varphi[z''/x], \theta[x \leftarrow z'']) \equiv_\alpha Subst(\varphi[z''/x], \theta)$ and similarly $Subst(\psi[z''/y], \theta[y \leftarrow z'']) \equiv_\alpha Subst(\psi[z''/y], \theta)$. Now by (IH) for point (5), we reduce what we need to prove to $Subst(\varphi[v/z], \theta) = Subst(\varphi[v/y], \theta)$, which holds by (IH) for point (9).
(10) Assume $\theta \restriction_{FV(\forall x.\varphi)} = \theta' \restriction_{FV(\forall x.\varphi)}$. In order to prove that $Subst(\forall x.\varphi, \theta) \equiv_\alpha Subst(\forall x.\varphi, \theta')$, note first that the variable z in the definition of substitution is the *same* in the two cases, and we need to show $\forall z.Subst(\varphi, \theta[x \leftarrow z]) \equiv_\alpha \forall z.Subst(\varphi, \theta'[x \leftarrow z])$, i.e., by (IH) for point (11), that $Subst(\varphi, \theta[x \leftarrow z]; [z'/z]) \equiv_\alpha Subst(\varphi, \theta'[x \leftarrow z]; [z'/z])$, i.e., that $Subst(\varphi, \theta[x \leftarrow z']) \equiv_\alpha Subst(\varphi, \theta'[x \leftarrow z'])$. The latter is true by (IH) for point (10), since $\theta[x \leftarrow z']\restriction_{FV(\varphi)} = \theta'[x \leftarrow z']\restriction_{FV(\varphi)}$.
(11) In order to prove that $Subst(\forall x.\varphi, \theta; \theta') \equiv_\alpha Subst(Subst(\forall x.\varphi, \theta), \theta')$, let z, z', z'' as in the definition of substitution (for each of the three involved substitutions). We need to show that $\forall z.Subst(\varphi, (\theta; \theta')[x \leftarrow z]) \equiv_\alpha \forall z''.Subst(Subst(\varphi[x \leftarrow z'], \theta), \theta'[z' \leftarrow z''])$, i.e., that $Subst(\varphi, (\theta; \theta')[x \leftarrow z])[z'''/z]] \equiv_\alpha Subst(Subst(\varphi[x \leftarrow z'], \theta), \theta'[z' \leftarrow z''])[z'''/z'']$, where $z''' \notin FV(\varphi) \cup \{z, z', z''\}$. Indeed, using (IH) for point (11) and the freshness of z, z', z'', z'''), we have the following chain of α-equivalencies and equalities:
$Subst(\varphi, (\theta; \theta')[x \leftarrow z])[z'''/z] \equiv_\alpha$
$Subst(\varphi, (\theta; \theta')[x \leftarrow z]; [z'''/z]) \equiv_\alpha$
$Subst(\varphi, (\theta; \theta')[x \leftarrow z''']) =$
$Subst(\varphi, \theta[x \leftarrow z']; \theta'[z' \leftarrow z''']) =$
$Subst(\varphi, \theta[x \leftarrow z']; \theta'[z' \leftarrow z'']; [z'''/z'']) \equiv_\alpha$
$Subst(\varphi, \theta[x \leftarrow z']; \theta'[z' \leftarrow z''])[z'''/z'']$,
which by (IH) for point (5) yield the desired result.
(12): The cases of logical connectors are obvious. Assume now $\varphi \equiv_\alpha \varphi'$. Then, by point (9), $\varphi[z/x] \equiv_\alpha \varphi'[z/x]$ for any x and z, in particular $\forall x.\varphi \equiv_\alpha \forall x.\psi$. □

THEOREM 1. *The Gentzen system $\mathcal{G}$ is sound and complete for generic first-order logic.*

Proof: Soundness: We need to check that the rules are sound. We only consider the quantifier rules, since the soundness of the others follows standardly. Let A be a model. For soundness of ($\forall$Left), it suffices that $A_{\forall x.\varphi} \subseteq A_{\varphi[T/x]}$, which is true because of the following: $\rho \in A_{\forall x.\varphi}$ is equivalent to $\rho[x \leftarrow a] \in A_\varphi$ for all $a \in A$, which implies $\rho[x \leftarrow A_T(\rho)] \in A_\varphi$, which in turn is equivalent, by Proposition 2.(2), to $\rho \in A_{\varphi[T/x]}$.

For ($\forall$Right), we shall tacitly use Proposition 2.(1,2) several times. Assume $\bigcap_{\chi\in\Gamma} A_\chi \subseteq \bigcup_{\psi\in\Delta} A_\psi \cup A_{\varphi[y/x]}$, where y is not free in $\Gamma, \Delta, \forall x.\varphi$. We need to show $\bigcap_{\chi\in\Gamma} A_\chi \subseteq \bigcup_{\psi\in\Delta} A_\psi \cup A_{\forall x.\varphi}$. For this, let $\rho \in \bigcap_{\chi\in\Gamma} A_\chi$ such that $\rho \notin \bigcup_{\psi\in\Delta} A_\psi$, and let us show that $\rho \in A_{\forall x.\varphi}$, i.e., that $\rho[x \leftarrow a] \in A_\varphi$ for all $a \in A$. Let $a \in A$. Because $\rho[y \leftarrow a]\restriction_{FV(\chi)} = \rho\restriction_{FV(\chi)}$ for each $\chi \in \Gamma \cup \Delta$ (since $y \notin FV(\chi)$), we have $\rho[y \leftarrow a] \in \bigcap_{\chi\in\Gamma} A_\chi$ and $\rho[y \leftarrow a] \notin \bigcup_{\psi\in\Delta} A_\psi$. Thus $\rho[y \leftarrow a] \in A_{\varphi[y/x]}$, i.e., $\rho[y \leftarrow a][x \leftarrow A_{\rho[y\leftarrow a]}(y)] \in A_\varphi$, i.e., $\rho[y \leftarrow a][x \leftarrow a] \in A_\varphi$. If $y = x$, we get $\rho[x \leftarrow a] \in A_\varphi$, as desired. On the other hand if $y \neq x$, then $\rho[y \leftarrow a][x \leftarrow a] = \rho[x \leftarrow a][y \leftarrow a]$, hence $\rho[x \leftarrow a][y \leftarrow a] \in A_\varphi$; and since $\rho[x \leftarrow a][y \leftarrow a]\restriction_{FV(\varphi)} = \rho[x \leftarrow a]\restriction_{FV(\varphi)}$, we get $\rho[x \leftarrow a] \in A_\varphi$, again as desired.

Completeness: The proof mainly follows a classical line (see [7]). Its only slightly specific part will be the one of constructing a model from a Hintikka signed set (sets H_{left} and H_{right} below).

Assume that $\Gamma \triangleright \Delta$ is a tautological sequent. Call a sequent *hopeless* if it is not an axiom and no rule can be applied backwards to it.[10] Note that a hopeless sequent $\Gamma' \triangleright \Delta'$ is one such that $\Gamma' \cap \Delta' = \emptyset$ and Γ', Δ' consist of atomic formulae; such a sequent is falsifiable, as shown below: let ρ be the identity valuation 1_{Var} in a Herbrand model that defines its relations as to make all formulae in Γ' true and all formulae in Δ' false; this is possible precisely because Γ' and Δ' are disjoint sets of atomic formulae.

As in [7], we construct backwards from $\Gamma \triangleright \Delta$ a possibly infinite proof tree, roughly by expanding the nodes not labelled with axioms or hopeless sequents via a fair application of rules to sequents. Special care needs to be taken when considering the ($\forall$Left) rule, since when the turn of this rule comes according to the considered fair scheduler, a counter n associated to the corresponding formula in Γ needs to be increased, and the rule needs to be applied for each of the first n terms (w.r.t., say, the lexicographic order). Moreover, dovetailing needs to be applied to the elements of Γ and Δ, provided these sets are infinite: fix an order on the set, and first consider the first element, then the first two elements etc. The (rather tedious) details are provided in [7] - the important thing is that these details are all independent of the concrete syntax of terms.

If the obtained tree is finite and all its leaves are labelled with axioms, the tree is completed, hence we have a proof for $\Gamma \triangleright \Delta$, as desired. If the tree is finite and there is a hopeless leaf falsifiable by some valuation ρ in a model M, then by the way rules in $\mathcal{G}$ were defined (to hold in an "iff" form), it follows that $\Gamma \triangleright \Delta$ itself is falsifiable by ρ in M. It remains to consider the case of an infinite tree. If we prove that $\Gamma \triangleright \Delta$ is falsifiable, then we are done. Consider the labels $\Gamma_i \triangleright \Delta_i$ of an arbitrarily chosen infinite path in the tree, with $\Gamma_=\Gamma$ and $\Delta_=\Delta$; by the construction of the tree, these labels do not include any axiom. If we set $H_{left} = \bigcup_{i\in I\!N} \Gamma_i$ and $H_{right} = \bigcup_{i\in I\!N} \Delta_i$, we get the following properties by the fair way in which rules were applied backwards in the construction of the tree:

- $\neg\varphi \in H_{left}$ implies $\varphi \in H_{right}$;
- $\neg\varphi \in H_{right}$ implies $\varphi \in H_{left}$;
- $\varphi \wedge \psi \in H_{left}$ implies $\varphi \in H_{left}$ and $\psi \in H_{left}$;
- $\varphi \wedge \psi \in H_{right}$ implies $\varphi \in H_{right}$ or $\psi \in H_{right}$;
- $\varphi \Rightarrow \psi \in H_{left}$ implies $\varphi \in H_{right}$ or $\psi \in H_{left}$;
- $\varphi \Rightarrow \psi \in H_{right}$ implies $\varphi \in H_{left}$ and $\psi \in H_{right}$;
- $\forall x.\varphi \in H_{left}$ implies for all $T \in Term$, $\varphi[T/x] \in H_{left}$;
- $\forall x.\varphi \in H_{right}$ implies there exists $y \in Var$ with $\varphi[y/x] \in H_{right}$ (in particular, there exists $T \in Term$ with $\varphi[T/x] \in H_{right}$).

[10] The proof trees are assumed to grow by backwards application of the rules, in the sense that a leaf is matched against a conclusion of the rule, and then the hypotheses of the rule are added to the tree.

Moreover, because the considered infinite path does not contain (nodes labelled with) axioms and because both the Γ_i's and the Δ_i's are totally ordered by inclusion w.r.t. their atomic formulae, it holds that $H_{left} \cap H_{right} \cap \mathit{Atomic\ Formulae} = \emptyset$.

In order to falsify $\Gamma \rhd \Delta$, it suffices to falsify $H_{left} \rhd H_{right}$. We define a Herbrand model A by letting $(T_1, \ldots, T_n) \in A_\pi$ iff $\pi(T_1, \ldots, T_n) \in H_{left}$, and take the valuation $\rho : \mathit{Var} \to A$ to be again 1_{Var}. We prove by structural induction on φ the following statement: [$\varphi \in H_{left}$ implies $A \models_\rho \varphi$] and [$\varphi \in H_{right}$ implies $A \not\models_\rho \varphi$]. If φ has the form $\pi(T_1, \ldots, T_n)$, then by the definition of A, $A \models_\rho \pi(T_1, \ldots, T_n)$ iff $\pi(T_1, \ldots, T_n) \in H_{left}$. In particular, $\pi(T_1, \ldots, T_n) \in H_{left}$ implies $A \models_\rho \pi(T_1, \ldots, T_n)$. Moreover, since $\pi(T_1, \ldots, T_n)$ is atomic and because of $H_{left} \cap H_{right} \cap \mathit{Atomic\ Formulae} = \emptyset$, it cannot happen that $\pi(T_1, \ldots, T_n) \in H_{right}$ and $A \models_\rho \pi(T_1, \ldots, T_n)$, thus $\pi(T_1, \ldots, T_n) \in H_{right}$ implies $A \not\models_\rho \pi(T_1, \ldots, T_n)$. The case of the logical connectives is straightforward. Assume now φ has the form $\forall x.\psi$. From $\forall x.\psi \in H_{left}$ we infer that $\psi[T/x] \in H_{left}$ for each term T (i.e., for each element T of A), and furthermore, by the induction hypotheses, that $A \models_\rho \psi[T/x]$, i.e., $\rho \in A_{\psi[T/x]}$, i.e., by Proposition 2.(2), $\rho[x \leftarrow A_\rho(T)] \in A_\psi$, i.e., $\rho[x \leftarrow T] \in A_\varphi$, for each $T \in A$; thus $A \models_\rho \forall x.\psi$. That $\forall x.\psi \in H_{right}$ implies $A \not\models_\rho \forall x.\psi$ can be proved similarly.

Thus $\Gamma \rhd \Delta$ is falsifiable and the proof is finished. □

THEOREM 2. *The Gentzen system $\mathcal{G}_=$ is sound and complete for generic first-order languages with equality.*

Proof: Soundness: We only check soundness of the rule regarding substitution. Let A and ρ such that $A_{T_1}(\rho) = A_{T_2}(\rho)$ for each $i \in \{1, n\}$. Then by point (c).(ii) in the definition of models, $A_{T[T'/x]}(\rho) = A_T(\rho[x \leftarrow A_{T'}(\rho)]) = A_T(\rho[x \leftarrow A_{T_1}(\rho)]) = A_{T[T_2/x]}(\rho)$.

Completeness: One can see that the effect of adding the equality rules amounts to adding the axioms *Eql* in the antecedent of sequents. Thus $\Gamma \rhd \Delta$ is provable in $\mathcal{G}_=$ iff $\Gamma' \cup \Gamma \rhd \Delta$ is provable in $\mathcal{G}$.

Now, given any model A in the language without equality (i.e., in the language that contains "=", but treats this symbol as just an ordinary binary relation symbol) that satisfies *Eql*, one defines the relation $\equiv$ by $a \equiv b$ iff $A \models_\rho x = y$ for some ρ with $\rho(x) = a$ and $\rho(y) = b$. Due to satisfaction of *Eql* by A, $\equiv$ is an equivalence compatible with the relations A_π and with the substitution, the latter in the sense that whenever $A_{T_1}(\rho) = A_{T_2}(\rho)$, it holds that $A_{T[T_1/x]}(\rho) = A_{T[T_2/x]}(\rho)$. Thus we can speak about a quotient model $A/\!\equiv$ and define, for each $\rho : \mathit{Var} \to A$, $\rho' : \mathit{Var} \to A/\!\equiv$ by $\rho'(x) = \rho(x)/\!\equiv$. Then a simple induction on φ shows that $A/\!\equiv \ \models_{\rho'} \varphi$ iff $A \models_\rho \varphi$. It follows that $\Gamma \rhd \Delta$ is tautological in the logic with equality iff $\Gamma' \cup \Gamma \rhd \Delta$ is tautological in the logic without equality.

Completeness of $\mathcal{G}_=$ now follows from completeness of $\mathcal{G}$. □

Below, by "proof tree" we mean "completed proof tree".

LEMMA 1. *Assume that $\mathcal{G}_E$ is closed under the rules (Drop-(e, a)). If $\Gamma \rhd \Delta_1 \cup \Delta_2$ is derivable in $\mathcal{G}_E$, then either $\Gamma \rhd \Delta_1$ or $\Gamma \rhd \Delta_2$ is derivable in $\mathcal{G}_E$.*

Proof: We prove the statement by induction on the size of a minimal proof tree for $\Gamma \rhd \Delta_1 \cup \Delta_2$.

If $\Gamma \rhd \Delta_1 \cup \Delta_2$ is an axiom, then $\Gamma \cap (\Delta_1 \cup \Delta_2) \neq \emptyset$, thus $\Gamma \cap \Delta_1 \neq \emptyset$ or $\Gamma \cap \Delta_1 \neq \emptyset$, making $\Gamma \rhd \Delta_1$ or $\Gamma \rhd \Delta_2$ an axiom.

If $\Gamma \rhd \Delta_1 \cup \Delta_2$ followed by an application of an (Inst-e) rule, then assume w.l.g. that $\Delta_2 = \Delta_2' \cup \{c(\overline{T})\}$ and $\Gamma \rhd \Delta_1 \cup \Delta_2$ followed from $\Gamma \cup \{a_i(\overline{z}, \overline{T})\} \rhd \Delta_1 \cup \Delta_2' \cup \{b_i(\overline{x}, \overline{T})\}$, with $i \in \{1, \ldots, n\}$. By the induction hypothesis, for each i, either $\Gamma a_i(\overline{z}, \overline{T}) \rhd \Delta_1$, or $\Gamma a_i(\overline{z}, \overline{T}) \rhd \Delta_2' b_i(\overline{x}, \overline{T})$. If the former is the case for at least one i, then since $\mathcal{G}_E$ is closed under (Drop-(e, a_i)), we get that $\Gamma \rhd \Delta_1$ is derivable. Otherwise $\Gamma a_i(\overline{z}, \overline{T}) \rhd \Delta_2' b_i(\overline{x}, \overline{T})$ for all i, hence, by the (Inst-e) rule, $\Gamma \rhd \Delta_2' c(\overline{T})$, i.e., $\Gamma \rhd \Delta_2$, is derivable. □

LEMMA 2. *$\mathcal{G}_E$ and $\mathcal{G}_E^1$ are equivalent (i.e., (Simple-Cut) can be eliminated from $\mathcal{G}_E^1$).*

Proof: We show that for each proof tree for $\Gamma \rhd \Delta$ having an application of the (Simple-Cut) rule only once, at the root, there exists a proof tree for $\Gamma \rhd \Delta$ that does not use this rule at all. Assume such a proof tree $\mathcal{T}$ starting with an application of (Simple-Cut) - then $\Gamma \rhd \Delta$ followed from $\Gamma \rhd d$ and $\Gamma d \rhd \Delta$.

If $\Gamma d \rhd \Delta$ is an axiom, then either $\Delta \cap \Gamma \neq \emptyset$, meaning $\Gamma \rhd \Delta$ is an axiom, or $d \in \Delta$, meaning that the proof tree of $\Gamma \rhd d$, which does not use (Simple-Cut) can be made into a proof tree of $\Gamma \rhd \Delta$ that does not use (Simple-Cut) either.

Assume now that $\Delta = \Delta' \cup \{c(\overline{T})\}$ and $\Gamma d \rhd \Delta$ followed from $\Gamma\, d\, a_i(\overline{z}, \overline{T}) \rhd \Delta' b_i(\overline{z}, \overline{T})$. We obtain a proof tree $\mathcal{T}'$ for $\Gamma \rhd \Delta$ by switching the applications of (Simple-Cut) and (Inst-e). More precisely, we derive $\Gamma \rhd \Delta$ from $\Gamma\, a_i(\overline{z}, \overline{T}) \rhd \Delta' b_i(\overline{z}, \overline{T})$, and each $\Gamma\, a_i(\overline{z}, \overline{T}) \rhd \Delta' b_i(\overline{z}, \overline{T})$ from $\Gamma a_i(\overline{z}, \overline{T}) \rhd d$ and $\Gamma\, d\, a_i(\overline{z}, \overline{T}) \rhd \Delta' b_i(\overline{z}, \overline{T})$, where the proof of $\Gamma a_i(\overline{z}, \overline{T}) \rhd d$ is copied from the one of $\Gamma \rhd d$. Applying the induction hypothesis for the proof trees (strictly smaller than $\mathcal{T}$) of $\Gamma\, d\, a_i(\overline{z}, \overline{T}) \rhd \Delta' b_i(\overline{z}, \overline{T})$, we can assume that they do not use (Simple-Cut), and we are done. □

PROPOSITION 5. *If $\mathcal{G}_E$ is closed under the rules (Drop-(e, a)), then it is also closed under (Cut), i.e., then $\mathcal{G}_E$ is equivalent to $\mathcal{G}_E^0$.*

Proof: Consider a proof tree of $\Gamma \rhd \Delta$ in $\mathcal{G}_E$, such that the (Cut) rule was applied only ones, at the root. Then $\Gamma \rhd \Delta$ followed from $\Gamma \rhd \Delta d$ and $\Gamma d \rhd \Delta$, where both latter sequents are derivable in $\mathcal{G}_E$. By Lemma 1, either $\Gamma \rhd \Delta$ or $\Gamma \rhd d$ is derivable in $\mathcal{G}_E$. The former case is precisely what we need to prove; in the latter case, $\Gamma \rhd d$, and thus $\Gamma \rhd \Delta$, are derivable in $\mathcal{G}_E^1$, i.e., by Lemma , in $\mathcal{G}_E$. □

THEOREM 3. *If $\mathcal{G}_E$ is closed under the rules (Drop-(e, a)), then $\mathcal{K}_E$ is complete for deducing E-tautological sequents $\Gamma \rhd d$, with Γ finite set of atomic formulae and d atomic formula.*

Proof: Follows at once from Proposition 5, noticing that in the system $\mathcal{G}_E$, if the sequent $\Gamma \rhd a$ is derivable, then all the sequents $\Gamma' \rhd \Delta'$ used in the proof tree have Δ' a singleton - in other words, $\mathcal{G}_E$ is a conservative extension of $\mathcal{K}_E$. □

PROPOSITION 6. *All the theories E in Subsection 3.1 are amenable, and thus $\mathcal{K}_E^=$ is complete for any of them.*

Proof: We only sketch a proof for the specification $\mathcal{SF}$ of System F. The cases of the other specifications, including their versions that contain declarations of constants of various types or kinds and equations that define their behavior, can be treated similarly, as they all need to "drop" atomic formulae like $\mathit{typeOf}(x, T)$ or $\mathit{kindOf}(t, K)$, with x and t fresh variables. The trick used in the below proof of restricting the "troubling rules" works in all these cases, because the troubling rules for all these specifications are the same, namely the ones that come from the equality axioms of transitivity, compatibility and substitution. After the restriction, a straightforward induction for proving closure under the drop rules works fine in each case.

Thus let us prove amenability of $\mathcal{SF}$. The only type of drop rules Drop-(e, a) come from e being [Abs] or (ξ) and $a(x, T)$ being $\mathit{typeOf}(x, T)$, with x fresh. Thus we need to prove closure under

$$\boxed{\frac{\Gamma\ \mathit{typeOf}(z,T) \rhd \Delta}{\Gamma \rhd \Delta}}$$

(with z fresh) of the following Gentzen system, where the notational conventions from the definition of $\mathcal{SF}$ apply, and where Γ and Δ denote, as usual, finite sets of atomic sentences:

$\dfrac{\cdot}{\Gamma \rhd \Delta}$ if $\Gamma \cap \Delta \neq \emptyset$	[Axiom]
$\dfrac{\Gamma\ \mathit{typeOf}(x,T) \rhd \Delta\ \mathit{typeOf}(X,T')}{\Gamma \rhd \Delta\ \mathit{typeOf}((\lambda x{:}T.X), T \to T')}$	[Inst-Abs]
$\dfrac{\Gamma \rhd \Delta\ \mathit{typeOf}(X,T)}{\Gamma \rhd \Delta\ \mathit{typeOf}(\lambda t.X, \Pi t.T)}$	[Inst-T-Abs]
$\dfrac{\Gamma \rhd \Delta\ \mathit{typeOf}(X, T \to T'),\ \Gamma \rhd \Delta\ \mathit{typeOf}(Y,T)}{\Gamma \rhd \Delta\ \mathit{typeOf}(XY,T')}$	[Inst-App]
$\dfrac{\Gamma \rhd \Delta\ \mathit{typeOf}(X, \Pi t.T)}{\Gamma \rhd \Delta\ \mathit{typeOf}(XT', T[t \leftarrow T'])}$	[Inst-T-App]
$\dfrac{\cdot}{\Gamma \rhd \Delta\ X = X}$	(Inst-Refl)
$\dfrac{\Gamma \rhd \Delta\ X = Y}{\Gamma \rhd \Delta\ Y = X}$	(Inst-Symm)
$\dfrac{\Gamma \rhd \Delta\ X = Y,\ \ \Gamma \rhd \Delta\ Y = Z}{\Gamma \rhd \Delta\ X = Z}$	(Inst-Trans)
$\dfrac{\cdot}{\Gamma \rhd \Delta\ T = T}$	(Inst-T-Refl)
$\dfrac{\Gamma \rhd \Delta\ T = T'}{\Gamma \rhd \Delta\ T' = T}$	(Inst-T-Symm)
$\dfrac{\Gamma \rhd \Delta\ T = T',\ \ \Gamma \rhd \Delta\ T' = T''}{\Gamma \rhd \Delta\ T = T''}$	(Inst-T-Trans)
$\dfrac{\Gamma \rhd \Delta\ X = Y,\ \ \Gamma \rhd \Delta\ T = T',\ \Gamma \rhd \Delta\ \mathit{typeOf}(X,T)}{\Gamma \rhd \Delta\ \mathit{typeOf}(Y,T')}$	(Inst-Comp$_{\mathit{typeOf}}$)
$\dfrac{\Gamma \rhd \Delta\ X = Y}{\Gamma \rhd \Delta\ Z[z \leftarrow X] = Z[z \leftarrow Y]}$	(Inst-Subst)
$\dfrac{\Gamma \rhd \Delta\ T = T'}{\Gamma \rhd \Delta\ T''[t \leftarrow T] = T''[t \leftarrow T']}$	(Inst-T-Subst)
$\dfrac{\Gamma\ \mathit{typeOf}(x,T) \rhd \Delta\ X = Y}{\Gamma \rhd \Delta\ \lambda x : T.X = \lambda x : T.Y}$	(Inst-ξ)
$\dfrac{\Gamma \rhd \Delta\ X = Y}{\Gamma \rhd \Delta\ \lambda t.X = \lambda t.Y}$	(Inst-T-ξ)
$\dfrac{\Gamma \rhd \Delta\ \mathit{typeOf}((\lambda x : T.X)Y, T')}{\Gamma \rhd \Delta\ (\lambda x : T.X)Y = X[x \leftarrow Y]}$	(Inst-β)
$\dfrac{\Gamma \rhd \Delta\ \mathit{typeOf}((\lambda t.X)T, T')}{\Gamma \rhd \Delta\ (\lambda t.X)T = X[t \leftarrow T]}$	(Inst-T-β)
$\dfrac{\Gamma \rhd \Delta\ \mathit{typeOf}(\lambda y : T.Xy, T')}{\Gamma \rhd \Delta\ X = \lambda y : T.Xy}$	(Inst-η)
$\dfrac{\Gamma \rhd \Delta\ \mathit{typeOf}(\lambda t.Xt, T')}{\Gamma \rhd \Delta\ X = \lambda t.Xt}$	(Inst-T-η)

The side-conditions of the above rules are the following:

- At [Inst-Abs] and (Inst-ξ), x does not occur (free) in $\Gamma \cup \Delta$;
- At [Inst-T-Abs] and (Inst-T-ξ), t does not occur in $\Gamma \cup \Delta$;
- At (Inst-η), $y \notin FV(X)$;
- At (Inst-T-η), $t \notin FV(X)$.

As expected, the above system of rules is obtained from $\mathcal{K}^{=}_{\mathcal{SF}}$ of Section 3.2 by adding finite sets Δ of atomic formulae to the succedents (here we write "*typeOf*" rather than ":" though).

Now, an argument for "$\Gamma \rhd \mathit{typeOf}(z,T'')\Delta$ (with z fresh) derivable implies $\Gamma \rhd \Delta$ derivable", inductive on the structure of a completed proof tree of $\Gamma \rhd \mathit{typeOf}(z,T'')\Delta$, *almost works*. For example, the induction step for rule [Inst-Abs] goes as follows: Assume $\Gamma \rhd \mathit{typeOf}(z,T'')\Delta$ was derived via [Inst-Abs] from $\Gamma\ \mathit{typeOf}(z,T'')\ \mathit{typeOf}(x,T) \rhd \Delta'\ \mathit{typeOf}(X,T')$, where $\Delta = \Delta'\ \mathit{typeOf}(\lambda x : T.X, T \to T')$. By the side-condition of [Inst-Abs], x is fresh for $\Gamma\ \mathit{typeOf}(z,T'')$, and thus z is different from x. Moreover, since $FV(\mathit{typeOf}(X.T')) \setminus FV(\mathit{typeOf}(\lambda x{:}T.X, T \to T')) \subseteq \{x\}$ and z is fresh for $\mathit{typeOf}(\lambda x : T.X, T \to T')$, it follows that z is fresh for $\mathit{typeOf}(X,T')$. We obtained z fresh both for $\Gamma\ \mathit{typeOf}(x,T)$ and for $\Delta'\ \mathit{typeOf}(X.T')$, and thus, by the inductive hypothesis, $\Gamma\ \mathit{typeOf}(x,T) \rhd \Delta'\ \mathit{typeOf}(X,T')$ is derivable, making $\Gamma \rhd \Delta'\ \mathit{typeOf}(\lambda x : T.X, T \to T')$, i.e., $\Gamma \rhd \Delta$, derivable, as desired.

The only problems are caused by the rules (Inst-Trans), (Inst-Comp$_{\mathit{typeOf}}$), and (Inst-Subst) - as they are, these rules cannot be considered in a straightforward induction, because they might have more data variables in their hypotheses than in their conclusion. However these rules can readily be replaced by apparently weaker rules, obtained from them by adding the following side-conditions:

- At (Inst-Trans): $FV(Y) \subseteq FV(X) \cup FV(Z) \cup FV(\Gamma)$;
- At (Inst-Comp$_{\mathit{typeOf}}$) : $FV(X) \subseteq FV(Y) \cup FV(\Gamma)$;
- At (Inst-Subst): $Z[z \leftarrow X]$ and $Z[z \leftarrow Y]$ are *syntactically* different.

Now the induction step works smoothly for these rules as well. □

LEMMA 3. *The forgetful mapping $(A, \_\langle\_\rangle, (A_T)_{T \in \Lambda(A)})$ to $(A, \_\langle\_\rangle, (A_T)_{T \in \Lambda})$ is a bijection, preserving satisfaction and each of the properties (P5),(P6), between pre-structures verifying (P1)-(P4) and simple pre-structures verifying (P1)-(P4).*

Proof: The inverse of the forgetful function maps simple pre-structures $(A, \_\langle\_\rangle, (A_T)_{T \in \Lambda})$ verifying (P1)-(P4) to $(A, \_\langle\_\rangle, (A_T)_{T \in \Lambda(A)})$, where for each term T in Λ, sequence of elements $a_1, \ldots, a_n$ in A and sequence of distinct variables $x_1, \ldots, x_n$, $A_{T[a_1/x_1, \ldots, a_n/x_n]}(\rho)$ is, by definition, $A_T(\rho[x_1 \leftarrow a_1, \ldots, x_n \leftarrow a_n])$. This definition is correct, because any term in $\Lambda(A)$ has the form $T[a_1/x_1, \ldots, a_n/x_n]$ for some T, and $A_T(\rho[x_1 \leftarrow a_1, \ldots, x_n \leftarrow a_n])$ does not depend on the choice of T. A simple induction on the structure of $\Lambda(A)$ terms shows that $A_{T[a_1/x_1, \ldots, a_n/x_n]}(\rho)$ is indeed equal to $A_T(\rho[x_1 \leftarrow a_1, \ldots, x_n \leftarrow a_n])$ in any pre-structure, and thus the two mappings are mutually inverse.

These mappings preserve satisfaction, since satisfaction is basically the same in each two structures related by these mappings - note also that in pre-structures only satisfaction of Λ-term equalities is defined. As for properties (P5) and (P6), one needs another induction to prove that in a pre-structure, verifying these properties w.r.t. Λ-terms is sufficient for them to hold w.r.t. $\Lambda(A)$-terms; here one uses again the equality $A_{T[a_1/x_1, \ldots, a_n/x_n]}(\rho) = A_T(\rho[x_1 \leftarrow a_1, \ldots, x_n \leftarrow a_n])$. □

LEMMA 4. *Each of the schemes (β), (η), (ext) is semantically equivalent in GFOL to its primed variant.*

Proof: Since (β'), (η') and (ext) are instances of the schemes (β), (η) and (ext), all we need to show is that the latter follow from the former; and this simply holds because in our logic it is sound to

infer $\varphi(T)$ from $\forall y.\varphi(y)$, and by the way substitution in formulae was defined. □

PROPOSITION 7. *The above two mappings are well defined and mutually inverse. Moreover, they preserve satisfaction and they can be restricted and corestricted to:*

(a) λ-models versus GFOL models satisfying $(\xi), (\beta)$;
(b) extensional λ-models versus GFOL models satisfying $(\xi), (\beta), (\eta)$.

Proof: We show that $L^{\#}$ is a GFOL model. Two of the GFOL model axioms, (c).(i) and (c).(iii), are precisely (P1) and (P3). The remaining axiom, (c).(ii), can be written as $A_{T[T_1/x_1,\ldots,T_n/x_n]}(\rho) = A_T(\rho[x_1 \leftarrow A_{T_1}(x_1), \ldots, A_{T_n}(x_n)])$. We check this by lexicographic induction on two criteria: the depth of T, and then the number n. The cases with T variable and T of the form $T'\,T''$ are simple, and they use (P1) and (P2). Assume now that T has the form $\lambda x.T'$. Since we work modulo α-equivalence, we can assume that x is not free in any of $T_1, \ldots, T_n$.

- If $x \notin \{x_1, \ldots, x_n\}$, then $T[\overline{T}/\overline{x}] = \lambda x.(T'[\overline{T}/\overline{x}])$. Thus we need to check $A_{\lambda x.(T'[\overline{T}/\overline{x}])}(\rho) = A_{\lambda x.T'}(\rho[\overline{x} \leftarrow A_{\overline{T}}(\rho)])$. By (P4), it is sufficient to consider $a \in A$ and prove $A_{T'[\overline{T}/\overline{x}]}(\rho[x \leftarrow a]) = A_{\lambda x.T'}(\rho[\overline{x} \leftarrow A_{\overline{T}}(\rho), x \leftarrow a])$, i.e., $A_{T'[\overline{T}/\overline{x}]}(\rho[x \leftarrow a]) = A_{\lambda x.T'}(\rho[x \leftarrow a][\overline{x} \leftarrow A_{\overline{T}}(\rho)])$, which is true by the induction hypothesis applied to T' and $\rho[x \leftarrow a]$.
- If $x \in \{x_1, \ldots, x_n\}$, say $x = x_1$, then $T[\overline{T}/\overline{x}] = T[T_2/x_2, \ldots, T_n/x_n]$ and by the induction hypothesis (second criterion), $T[T_2/x_2, \ldots, T_n/x_n] = A_T(\rho[x_2 \leftarrow A_{T_2}(x_1), \ldots, x_n \leftarrow A_{T_n}(x_n)]$. Finally, because $x \notin FV(T)$, the valuations $\rho_1 = \rho[\overline{x} \leftarrow A_{\overline{T}}(\rho)]$ and $\rho_2 = \rho[x_2 \leftarrow A_{T_2}(x_1), \ldots, x_n \leftarrow A_{T_n}(x_n)]$ coincide on $FV(T)$, hence, by (P3), $A_T(\rho_1) = A_T(\rho_2)$ and we are done.

(Above, we used the obvious tuple notations $\overline{x}$ for $(x_1, \ldots, x_n)$, $\overline{T}$ for $(T_1, \ldots, T_n)$ etc.) $L^{\#}$ satisfies (ξ) because (P4) is nothing else but a semantic statement of (ξ).

We show that $M^{\$}$ is a λ-model. Properties (P1) and (P3) are required for generic models as well, hence they hold. (P2) holds as an instance of the axiom (c).(ii) of GFOL models: $A_{T_1\,T_2}(\rho) = A_{x_1x_2[T_1/x_1,T_2/x_2]}(\rho) = A_{x_1x_2}([\rho[x_1 \leftarrow A_{T_1}(\rho), x_2 \leftarrow A_{T_2}(\rho)]) = A_{T_1}(\rho)\langle A_{T_2}(\rho)\rangle$. (We also used the definition of $\_\langle\_\rangle$.) Again, (P4) holds in $M^{\$}$ because M satisfies (ξ).

That $\#$ and $\$$ are mutually inverse follows from the fact that $a\langle b\rangle = A_{xy}(\rho)$, with ρ mapping x to a and y to b, holds in any pre-structure verifying (P1)-(P4).

In order to see that the pre-structure is a λ-model (i.e., it also verifies (P5) iff the corresponding GFOL model satisfies (β), note that (P5) is just a semantic statement of (β'), which is equivalent to (β) by Lemma 4. Similarly, extensional λ-models correspond to $(\beta) \cup (\eta)$-GFOL models because of the following:

- under the $(\beta), (\xi)$ assumptions, (η) is equivalent to (ext);
- (ext) is equivalent to (ext') by Lemma 4;
- (ext') is just a semantic statement of (P6).

Finally, both $\#$ and $\$$ preserve satisfaction since it has the same definition for equations in pre-structures and generic models. □

PROPOSITION 8. *For all typing judgements $\Gamma \triangleright X : T$, $\vdash_{SF} \Gamma \triangleright X : T$ iff $\mathcal{SF} \vdash^{=}_{GFOL} \Gamma^{\#} \Rightarrow typeOf(X, T)$.*[11]

Proof: Let $\mathcal{TSF}$ denote the typing fragment of $\mathcal{SF}$, i.e., the one consisting of axioms whose labels were enclosed into brackets: [Abs], [T-Abs], [App], [T-App]. $\mathcal{SF}$ was proved amenable at Proposition 6 by showing that $\mathcal{G}^{=}_{\mathcal{SF}}$ is closed under the drop rules; the proof was performed by induction on the structure of a proof tree in $\mathcal{G}^{=}_{\mathcal{SF}}$, by considering an induction step for each of its rules; this proof works unchanged for the strict subset of these rules forming $\mathcal{G}_{\mathcal{TSF}}$, and thus we obtain that $\mathcal{G}_{\mathcal{TSF}}$ is closed under the drop rules, making $\mathcal{K}_{\mathcal{TSF}}$ complete by Theorem 3. Moreover, the rules of $\mathcal{K}_{\mathcal{TSF}}$ can be seen to coincide with those of the System-F's original typing, i.e., the $\mathcal{K}_{\mathcal{TSF}}$'s [Axiom], [Inst-Abs], [Inst-T-Abs], [Inst-App], and [Inst-T-App] are nothing but the System-F's [SF-InVar], [SF-Abs], [SF-T-App], [SF-T-Abs], and [Inst-T-App], modulo the notation $x\!:\!T$ for $typeOf(x, T)$. We thus obtain $\vdash_{SF} \Gamma \triangleright X : T$ iff $\mathcal{TSF} \vdash_{\text{GFOL}} \Gamma^{\#} \Rightarrow typeOf(X, T)$ for all typing judgements $\Gamma \triangleright X : T$. Finally, notice that the axioms of $\mathcal{SF}$ others than those of $\mathcal{TSF}$ do not affect typing, i.e., $\mathcal{SF} \vdash^{=}_{\text{GFOL}} \Gamma^{\#} \Rightarrow typeOf(X, T)$ iff $\mathcal{TSF} \vdash_{\text{GFOL}} \Gamma^{\#} \Rightarrow typeOf(X, T)$. □

LEMMA 5. *Henkin and 1-Henkin models are equivalent, in that there exists a satisfaction- preserving and reflecting surjection $\&_1$ between the class of Henkin models and that of 1-Henkin models. That is to say: for all Henkin models H and well-typed equation $\Gamma \triangleright X = Y : T$,*

$$H \models \Gamma \triangleright X = Y : T \quad \textit{iff} \quad H^{\&_1} \models \Gamma \triangleright X = Y : T.$$

Proof: The mapping $\&_1$ makes disjoint copies of the $\mathcal{D}om_\sigma$'s whenever necessary, curries the functions $App_{\sigma,\tau}$ and App_f, and deletes from $\mathcal{F}$ all functions that do not come from terms. $\&_1$ preserves and reflects satisfaction because satisfaction only considers functions in $\mathcal{F}$ that come from terms, and hence it relays on *the same structure* for H as for $H^{\&_1}$. $\&_1$ is a surjection because any Henkin model with disjoint domains $\mathcal{D}om_\sigma$ and all functions coming from terms yields a 1-Henkin model by uncurring. □

LEMMA 6. *1-Henkin and 2-Henkin models are equivalent, in that there exist two satisfaction- preserving and reflecting mappings $\&^1_2$ and $\&^2_1$ between the two classes of models.*

Proof: $\&^1_2$ maps a 1-Henkin model to a 2-Henkin model by forgetting $\mathcal{F}$ and Π and by defining $App_{H_{\Pi t.T}(\gamma)}$ to be $App_{\tau \mapsto H_T(\gamma[\tau \leftarrow t])}$; the definition is correct by property (6) in the definition of 1-Henkin models. Conversely, $\&^2_1$ maps a 2-Henkin model to a 1-Henkin model by defining $\mathcal{F}$ as at point (c) in the definition of 1-Henkin models, and App_f by $App_{H_{\Pi t.T}(\gamma)}$ if f has the form $\tau \mapsto H_T(\gamma[t \leftarrow \tau])$. Satisfaction is seen to be precisely *the same* in corresponding 1-Henkin and 2-Henkin models. □

LEMMA 7. *2-Henkin and 3-Henkin models are equivalent, in that there exist two satisfaction- preserving and reflecting mappings $\&^2_3$ and $\&^3_2$ between the two classes of models.*

Proof: Let H be a 2-Henkin model. The 3-Henkin model $\&^2_3(H)$ is defined as follows: $\mathcal{D} = \bigcup \mathcal{D}om$. $App(d, d') = App_{\tau,\sigma}(d, d')$ if there exist τ, σ such that $d \in \mathcal{D}om_\tau$ and $d' \in \mathcal{D}om_\sigma$ and $App(d, d')$ arbitrary otherwise; the definition is correct because, thanks to disjointness of the $\mathcal{D}om_\tau$'s and injectivity of $\rightarrow$, there can be at most one pair (τ, σ) as above. Similarly, $TApp(d, \tau) = App_{H_{\Pi t.T}(\gamma)}(d, \tau)$ if there exist t, T, γ such that $d \in \mathcal{D}om_{H_{\Pi t.T}(\gamma)}$. The relation $typeOf$ is the following: $typeOf(d, \tau)$ iff $d \in \mathcal{D}om_\tau$. Everything else remains the same.

Conversely, let H be a 3-Henkin model. The 2-Henkin model $\&^3_2(H)$ is defined as follows: $\mathcal{D}om_\sigma = \{d \in \mathcal{D} : typeOf(d, \sigma)\}$. $App_{\tau,\sigma}$ is the restriction and corestriction of App to $\mathcal{D}om_{\tau\rightarrow\sigma} \times \mathcal{D}om_\tau \rightarrow \mathcal{D}om_\sigma$; the correctness of this definition is ensured by property (2) in the definition of 3-Henkin models. Similarly, $TApp_{H_{\Pi t.T}(\gamma)}$ is the restriction and corestriction of $TApp$ to $\mathcal{D}om_{H_{\Pi t.T}(\gamma)} \times \mathcal{T} \rightarrow \mathcal{D}$, with correctness ensured by property (3)

[11] Recall that we identify GFOL-formulae with their universal closures.

in the definition of 3-Henkin models. Everything else remains the same.

Note that $\&_2^3 \circ \alpha_3^2$ is the identity mapping, thus 2-Henkin models are somehow more "concise" than 3-Henkin models. Again, there is nothing to prove about preservation an reflection of satisfaction, since again the satisfaction relation is the same in two corresponding models. □

LEMMA 8. *Let H be a 3-Henkin model, $\gamma : TVar \to \mathcal{T}$ and $\delta : DVar \to \mathcal{D}$. Then there for any two pairs (Γ, T) and (Γ', T') such that $\vdash \Gamma \rhd X : T$ and $\vdash \Gamma' \rhd X : T'$ such that $H_{\Gamma \rhd X:T}$ and $H_{\Gamma' \rhd X:T'}$ are defined on (γ, δ), it holds that $H_{\Gamma \rhd X:T}(\gamma, \delta) = H_{\Gamma' \rhd X:T'}(\gamma, \delta)$.*

Proof: Easy induction on the derivation of typing judgements; the idea is that all the information needed for interpreting X is already in the valuation (γ, δ), and Γ and T can only "confirm" this information. □