Term-Generic Logic*

Andrei Popescu** and Grigore Roşu

Department of Computer Science,
University of Illinois at Urbana-Champaign.
{popescu2,grosu}@cs.uiuc.edu

Abstract. *Term-generic logic (TGL)* 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. TGL has a complete Gentzen system generalizing that of first-order logic. A certain fragment of TGL, called HORN^2, possesses a much simpler Gentzen system, similar to traditional typing derivation systems of λ-calculi. HORN^2 appears to be sufficient for defining a whole plethora of λ-calculi as *theories* inside the logic. Within intuitionistic TGL, a HORN^2 specification of a calculus is likely to be *adequate by default*. A bit of extra effort shows adequacy w.r.t. classic TGL as well, endowing the calculus with a complete loose semantics.

1 Introduction

First-order logic (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, the *free* ones, 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 *term-generic logic (TGL)*, as a first-order logic parameterized by any terms that come with abstract notions of *free variable* and *substitution*. More precisely, in TGL 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, satisfying some expected properties. TGL models provide interpretations of terms that satisfy, again, some reasonable properties. We show that TGL

* Supported in part by NSF grants CCF-0448501, CNS-0509321 and CNS-0720512, by NASA contract NNL08AA23C, by the Microsoft/Intel funded Universal Parallel Computing Research Center at UIUC, and by several Microsoft gifts.

** Also: Institute of Mathematics "Simion Stoilow" of the Romanian Academy.

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 and substitution.

Because of not committing to any particular definition of term, TGL can be instantiated to different types of terms, such as standard FOL terms or different categories of (typed or untyped) λ-terms. When instantiated to standard FOL terms, TGL becomes, as expected, precisely FOL. However, when instantiated to more complex terms, e.g., the terms of λ-calculus, TGL becomes a logic where a particular calculus is a particular theory. For example, the TGL axiom for typing abstractions in simply-typed λ-calculus can be

$$(\forall x.\, x : t \Rightarrow X : t') \Rightarrow (\lambda x{:}t.X) : t \to t'$$

where x and t, t' denote data and type variables, respectively, X denotes an arbitrary data term, $\Rightarrow$ is the logical implication, and $\to$ is the arrow type construct (binary operator on types). The above is an axiom-scheme, parameterized by any choice of variables x, t, t' and term X (and, as customary, each of its instances is implicitly assumed universally quantified over all its free variables). The colons in $x : t$ and $X : t'$ and the outermost colon in $(\lambda x : t.\, X) : t \to t'$ refer to a binary relation symbol in TGL, while the colon in $\lambda x : t.X$ is part of the term syntax. The term X may contain the free variable x, which is bound by $\forall$ in the lefthand side of the outermost implication, and by λ in the righthand side. Both these capturings of x from X are *intended* – in fact, the migration of x between the two scopes is at the heart of the intended typing mechanism: x is an *actual*, but *arbitrary* input to the function described by X in the former case, and a *formal* parameter in the latter; the type $t \to t'$ is assigned to the abstraction $\lambda x : t.X$ by "experimenting" with arbitrary x's of type t and "observing" if the result has type t'. (Using the same notation for actual as for formal parameters of functional expressions is well-established mathematical practice.)

A possible instance of the above axiom-scheme, taking, e.g., $\lambda y : t''.\, y\, x$ as X and spelling out all the universal quantifications, is

$$\forall t, t', t''.\ (\forall x.\, x : t \Rightarrow (\lambda y : t''.\, y\, x) : t') \Rightarrow (\lambda x : t.\, \lambda y : t''.\, y\, x) : t \to t',$$

which implies in TGL, instantiating t'' with $t \to t'''$ and t' with $(t \to t''') \to t'''$,

$$\forall t, t'''.\ (\forall x.\, x : t \Rightarrow (\lambda y : t \to t'''.\, y\, x) : (t \to t''') \to t''') \Rightarrow$$
$$(\lambda x : t.\, \lambda y : t \to t'''.\, y\, x) : t \to (t \to t''') \to t'''.$$

Moreover, we can prove in TGL, using again the above axiom-scheme and another axiom for application, that the hypothesis (i.e., the lefthand side of the outermost $\Rightarrow$) in the latter sentence is true for all t', t''', hence we obtain a TGL derivation of $\forall t, t'''.\, (\lambda x : t.\, \lambda y : t \to t'''.\, y\, x) : t \to (t \to t''') \to t'''$.

A specification of a calculus in TGL 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, TGL models are initially "blank", in that they are only required to interpret the terms consistently with substitution – it is the axioms that customize the models. For instance, the previously discussed description of x as an "actual, but arbitrary parameter" is not merely an informal idea to help the intuition, but a mathematical fact

within the TGL semantics: when "escaped" from the scope of λ into the scope of $\forall$, x indeed denotes an actual, but arbitrary inhabitant of a desirable model.

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 [5], that simply "replaces equals by equals", thus avoiding the more intricate first-order proofs. An important goal of this paper is to also investigate conditions under which sublogics of TGL admit specialized coarse-granularity proof systems.

It appears that a certain fragment of TGL, that we call HORN^2, is sufficient for calculi-specification purposes. HORN^2 consists of TGL 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 (single-hypothesis)[1] Horn implications. We show that, under a reasonable 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

$$\frac{\Gamma, a_i(\overline{z},\overline{T}) \triangleright b_i(\overline{z},\overline{T}) \text{ for all } \ i \in \{1,\ldots,n\}}{\Gamma \triangleright c(\overline{T})}$$

where $\overline{T}$ is a tuple of terms substituting $\overline{y}$ and $\overline{z}$ is a fresh tuple of variables substituting $\overline{x}$. The (multiple-formulae antecedent, single-formula succedent) structure of this system follows the style of intuitionistic logic, and indeed we show that it specializes the Gentzen system of the intuitionistic version of TGL. Thus we obtain for the HORN^2 fragment an intuitionistic proof system which is complete w.r.t. classical models! Moreover, this "lower-level" Gentzen system, extracted from the higher-level notation used in the HORN^2 theory, recovers the original calculus itself (bringing what in syntactic encoding frameworks is usually referred to as *adequacy* of the representation). For instance, the HORN^2 deduction rule corresponding to the aforementioned typing axiom for typed λ-calculus is *precisely* the familiar context-based typing rule for abstractions:

$$\frac{\Gamma, z{:}T \triangleright Z : T'}{\Gamma \triangleright (\lambda z{:}T.\, Z) : T \to T'}[z \text{ fresh for } \Gamma]$$

By substitution, x from the typing axiom became a fresh z in the deductive system, the variables t, t' became arbitrary terms T, T', and X became a term Z such that the positions in which x occurred in X are the same as the positions in which z now occurs in Z (because the term X and the positioning of x in X were arbitrary in the typing axiom, it follows that the term Z and the positioning of

[1] Single hypothesis, in the sense that each $a_i(\overline{x},\overline{y})$ has to be an atomic formula, as opposed to being a conjunction of atomic formulae.

z in Z are also arbitrary in the resulted deduction rule). This transformation is prescribed uniformly, i.e., calculus-independently, for any HORN[2] theory.

The remainder of this paper is structured as follows. Section 2 introduces classic TGL (syntax, models, institutional structure, Gentzen system and completeness theorem) and intuitionistic TGL. Section 3 discusses the HORN[2] fragment and its specialized Gentzen systems, whose completeness "prepares" the logic for future adequacy results. Section 4 illustrates the TGL *adequate by default* specification style for λ-calculi, taking System F as a working example. Section 5 discusses related work and draws conclusions. More details regarding the topics addressed in this paper, including proofs of the stated facts, can be found in the technical report [23] – the main part of the report has the same content as this paper, while the appendix of the report contains further details and proofs.

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 a development of first-order logic that does not depend on the syntax of terms, but *only on the properties of substitution*. We first develop the logic in an unsorted form and without equality, and later sketch equality and order-sorted extensions, as well as an intuitionistic variant.

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 $FV\colon Term \to \mathcal{P}_f(Var)$ *(where* $\mathcal{P}_f$ *means "the set of finite sets of"); the elements of* $FV(T)$ *are called* free variables*, or simply* variables*, of* T*;*
(c) A mapping $Subst\colon Term \times Term^{Var} \to Term$*, called* substitution*.*

These are subject to the following requirements (where x *ranges over variables,* T, T' *over terms, and* θ, θ' *over maps in* $Term^{Var}$*):*

(1) $Subst(x, \theta) = \theta(x)$*;*
(2) $Subst(T, Var \hookrightarrow Term) = T$*;*
(3) If $\theta{\restriction_{FV(T)}} = \theta'{\restriction_{FV(T)}}$*, then* $Subst(T, \theta) = Subst(T, \theta')$*;*[2]
(4) $Subst(Subst(T, \theta), \theta') = Subst(T, \theta; \theta')$*, where* $\theta; \theta' : Var \to Term$ *is defined by* $(\theta; \theta')(y) = Subst(\theta(y), \theta')$*;*
(5) $FV(x) = \{x\}$*;*
(6) $FV(Subst(T, \theta)) = \bigcup\{FV(\theta(x)) : x \in FV(T)\}$*.*

Note that we assume the notion of term coming together with a notion of substitution which is *composable* (condition (4) above). In general, for a syntax with bindings, composability of substitution does not hold for raw terms, but only for α-equivalence classes – therefore, in the concrete instances of our logic to calculi

[2] Here and later, if $f : U \to V$ and $U' \subseteq U$, $f{\restriction_{U'}}$ denotes the restriction of f to U'.

with bindings, TGL terms will be α-equivalence classes of what are usually called (raw) "terms" in these calculi. Conditions (1)-(6) from Definition 1 are natural (and well-known) properties of substitution holding for virtually all notions of terms with static binding (modulo α-equivalence).

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 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 $Subst(T, [T_1/x_1, \ldots, T_n/x_n])$.

Definition 2. *A* term-generic language *consists of a term syntax* $(Term, FV, Subst)$ *over a set* Var *and an at most countable ranked set* $\Pi = (\Pi_n)_{n \in \mathbb{N}}$*, of* relation symbols*. A TGL* model *for a language as above is a triple of the form* $(A, (A_T)_{T \in Term}, (A_{(n,\pi)})_{n \in \mathbb{N}, \pi \in \Pi_n})$*, where:*

(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 the following hold for all* $x \in Var$*,* $T \in Term$*,* $\rho, \rho' \in A^{Var}$*, and* $\theta \in Term^{Var}$*:*
- *(b.i)* $A_x(\rho) = \rho(x)$*;*
- *(b.ii) If* $\rho\restriction_{FV(T)} = \rho'\restriction_{FV(T)}$*, then* $A_T(\rho) = A_T(\rho')$*;*
- *(b.iii)* $A_{Subst(T,\theta)}(\rho) = A_T(A_\theta(\rho))$*, where* $A_\theta : A^{Var} \to A^{Var}$ *is defined by* $A_\theta(\rho)(y) = A_{\theta(y)}(\rho)$*.*

(c) For each $n \in \mathbb{N}$ *and* $\pi \in \Pi_n$*,* $A_{(n,\pi)}$ *is an* n*-ary relation on* A*.*

Thus, unlike in classic FOL models where the interpretation of terms is *built* from operations, in TGL models the interpretation of terms is *assumed* (in the style of Henkin models). It turns out that condition (b.ii) is redundant (follows from the other conditions and Definition 1 – see Section F.1 in [23] for a proof)[3] – we keep it though as part of the definition of a model for the sake of symmetry with Definition 1.

In what follows, we let x, x_i, y, u, v, etc., range over variables, T, T_i, T', etc., over terms, θ, θ', etc., over maps in $Term^{Var}$, ρ, ρ', etc., over valuations in A^{Var}, and π, π', etc., over relation symbols. Sometimes we simply write $Term$ for term syntaxes $(Term, FV, Subst)$ and $(Term, \Pi)$ for term-generic languages.

Formulae are defined as usual, starting from *atomic formulae* $\pi(T_1, \ldots, T_n)$ and applying connectives $\wedge, \Rightarrow$ and quantifier $\forall$. (Other connectives and quantifiers may of course be also considered, but we omit them since they shall not be needed for our specifications in this paper.) 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 $(A_{T_1}(\rho), \ldots, A_{T_n}(\rho)) \in A_\pi$; $\rho \in A_{\varphi \Rightarrow \psi}$ iff $\rho \in A_\varphi$ implies $\rho \in A_\psi$; $\rho \in A_{\varphi \wedge \psi}$ 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, $A \models \Gamma$ means $A \models \varphi$ for all $\varphi \in \Gamma$. Above, and from now on, we let φ, ψ, χ range over formulae and A, B over models (sometimes, when we want to distinguish models from their carrier set A, B, we write $\mathcal{A}, \mathcal{B}$ for the models). For formulae, the notions of free variables, α-equivalence, and substitution are the natural ones, defined similarly to the case of FOL, but

[3] We are indebted to one of the referees for bringing this to our attention.

on top of our generic terms rather than FOL terms. For substitution in formulae we adopt notational conventions similar to the ones about substitution in terms, e.g., $\varphi[T/x]$. Note that TGL is a logic generic only w.r.t. terms - formulae are "concrete" first-order formulae built over generic terms, with a "concrete" (and not generic) notion of α-equivalence, standardly defined using the bindings from quantifiers, which preserves satisfaction and the free variables and is compatible with substitution and the language constructs. Hereafter we identify formulae modulo α-equivalence. Let $\overline{x} = (x_1, \ldots, x_n)$ be a tuple of variables and $J = \{y_1, \ldots, y_m\}$ a set of variables. Then $\mathit{Vars}(\overline{x})$ denotes the set $\{x_1, \ldots, x_n\}$, $\forall \overline{x}.\, \varphi$ denotes $\forall x_1 \ldots \forall x_n . \varphi$, and $\forall J.\, \varphi$ denotes $\forall y_1 \ldots \forall y_m . \varphi$ (the latter notation making an (immaterial for our purposes) choice of a total ordering on J). A *sentence* is a formula with no free variables. The *universal closure* of a formula φ is the sentence $\forall FV(\varphi).\, \varphi$. (See Section A.1 in [23] for more details.)

The inclusion of an emphasized equality symbol in our logic, interpreted in all models as equality, yields *TGL with equality*. *Many-sorted* and *order-sorted* variants of TGL (in the style of [13]) can also be straightforwardly obtained, by extending Definition 1 to term syntaxes involving multiple sorts (syntactic categories) and Definition 2 to models having as carriers sort-indexed sets. For example, in the case of order-sorted TGL, a poset $(S, <)$ of sorts is fixed and carriers of models are families of sets $(A_s)_{s \in S}$ such that $s < s'$ implies $A_s \subseteq A_{s'}$ for all $s, s' \in S$. (See Sections A.2 and A.3 in [23] for details.) All the concepts and results about TGL 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.

FOL. As expected, classic FOL is an instance of TGL. Indeed, let $(\mathit{Var}, \Sigma, \Pi)$ be a classic first-order language, where $\Sigma = (\Sigma_n)_{n \in \mathbb{N}}$ 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 $FV : \mathit{Term} \to \mathcal{P}_f(\mathit{Var})$ giving *all* the variables in each term and $\mathit{Subst} : \mathit{Term} \times \mathit{Term}^{\mathit{Var}} \to \mathit{Term}$ the usual substitution on FOL terms. Define a term-generic language as (Term, Π). A classic FOL model $(A, (A_\sigma)_{\sigma \in \Sigma}, (A_{(n,\pi)})_{n \in \mathbb{N}, \pi \in \Pi_n})$ yields a TGL model $(A, (A_T)_{T \in \mathit{Term}}, (A_{(n,\pi)})_{n \in \mathbb{N}, \pi \in \Pi_n})$ by defining the meaning of terms as derived operations. Conversely, a TGL model $(A, (A_T)_{T \in \mathit{Term}}, (A_{(n,\pi)})_{n \in \mathbb{N}, \pi \in \Pi_n})$ yields an 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 two model mappings are mutually inverse and preserve satisfaction. Thus, for this particular choice of terms, TGL yields FOL.

A Formula-Typed Logic (an "Extremely-Typed" λ-Calculus). FOL is a simple instance of TGL. However, TGL terms may be arbitrarily exotic. Besides terms of λ-calculi (discussed in Section 4), one may also have terms that interfere with formulae in non-trivial ways, where, e.g., terms may abstract variables having formulae as types (thinking of types as sets defined by comprehension). Let $(\mathit{Var}, \Sigma, \Pi)$ be a classic first-order language and:

$\mathit{Term} \ ::= \ \mathit{Var} \mid \Sigma(\mathit{Term}, \ldots, \mathit{Term}) \mid \mathit{Term}\ \mathit{Term} \mid \lambda\, \mathit{Var} : \mathit{Fmla}.\ \mathit{Term}$

$\mathit{Fmla} \ ::= \ \Pi(\mathit{Term}, \ldots, \mathit{Term}) \mid \mathit{Fmla} \Rightarrow \mathit{Fmla} \mid \mathit{Fmla} \wedge \mathit{Fmla} \mid \forall\, \mathit{Var}.\, \mathit{Fmla}$,

with the natural restrictions w.r.t. the rank of operations/relations. Free variables and substitution are as expected, making terms up to α-equivalence a TGL term syntax. Moreover, although formulae and terms were defined mutually recursively, the former are still nothing but first-order formulae over terms, hence fall under TGL. Does the interpretation of formulae inside terms match their first-order interpretation at the top level? The answer is "no, unless axioms require it" (remember that TGL models are blank, but customizable). Here, postulating $(\forall x.\,(\varphi \Leftrightarrow \psi) \wedge (\varphi \Rightarrow T = T')) \Rightarrow \lambda x\!:\!\varphi.T = \lambda x\!:\!\psi.T'$ does the job.

The TGL institution. Next we submit TGL to a standard well-behaving test for a logical system, organizing it as an institution [12]. By doing so, we present TGL terms and models in a more structural light, and, more importantly, create a framework for λ-calculi with different flavors to cohabitate with each other and with classic FOL *under different signatures and axioms, but within the same logic*, connected through the railway system of signature morphisms.

Let Var be a fixed set of variables. The *signatures* are term-generic languages $(Term, \Pi)$ over Var. The $(Term, \Pi)$- *sentences*, *models* and *satisfaction relation* were already defined for TGL. Given $\theta : Var \to Term$, we write $\overline{\theta}$ for the map $T \mapsto Subst(T, \theta)$. Moreover, for any model $(A, (A_T)_{T \in Term}, (A_{(n,\pi)})_{n \in \mathbb{N}, \pi \in \Pi_n})$ and map $\rho : Var \to A$, we write $\overline{\rho}^A$ for the map $T \mapsto A_T(\rho)$. A model can be alternatively presented as a tuple $\mathcal{A} = (A, \overline{\cdot}^A, (A_{(n,\pi)})_{n \in \mathbb{N}, \pi \in \Pi_n})$ where the $A_{(n,\pi)}$'s are relations as before and $\overline{\cdot}^A : A^{Var} \to A^{Term}$ is such that $\overline{\rho}^A \circ (Var \hookrightarrow Term) = \rho$, $[\overline{\rho}^A(T) = \overline{\rho'}^A(T)$ whenever $\rho \restriction_{FV(T)} = \rho' \restriction_{FV(T)}]$, and $\overline{\rho}^A \circ \overline{\theta} = \overline{\overline{\rho}^A \circ \theta}^A$. A *model homomorphism* between $\mathcal{A} = (A, \overline{\cdot}^A, (A_{(n,\pi)})_{n \in \mathbb{N}, \pi \in \Pi_n})$ and $\mathcal{B} = (B, \overline{\cdot}^B, (B_{(n,\pi)})_{n \in \mathbb{N}, \pi \in \Pi_n})$ is a map $h : A \to B$ that commutes with the relations in the usual way and has the property that $h \circ \overline{\rho}^A = \overline{h \circ \rho}^B$ for all $\rho \in A^{Var}$. (Note the structural similarity between the conditions defining the three concepts of term syntax, model and model homomorphism, which allows one to easily see that $(Term, \overline{\cdot}, (Term_{(n,\pi)})_{n \in \mathbb{N}, \pi \in \Pi_n})$ is a model for any choice of relations $Term_{(n,\pi)}$ and that $(Term, \overline{\cdot}, (\emptyset)_{\pi \in \Pi})$ is freely generated by Var.)

A *signature morphism* between $(Term, \Pi)$ and $(Term', \Pi')$ is a pair (u, v) with $v = (v_n : \Pi_n \to \Pi'_n)_{n \in \mathbb{N}}$ and $u : Term \to Term'$ such that $u \circ (Var \hookrightarrow Term) = (Var \hookrightarrow Term')$, $FV(u(T)) = FV(T)$ for all $T \in Term$ and $u \circ \overline{\theta} = \overline{u \circ \theta}' \circ u$ (where $\overline{\cdot}'$ is the map $Term' \to Term'$ associated to the term syntax $Term'$) for all $\theta : Var \to Term$. (Intuition for the last condition on signature morphism: say we have concrete terms, like the ones of FOL or λ-calculus, θ maps x to S and all other variables to themselves (thus $\overline{\theta}(T) = T[S/x]$ for all terms T), and u maps each T to the term $T[g/f]$ obtained by replacing an operation symbol f with an operation symbol g of the same arity; then one has $T[S/x][g/f] = T[g/f][S[g/f]/x]$, i.e., $(u \circ \overline{\theta})(T) = (\overline{u \circ \theta}' \circ u)(T)$, for all terms T.) To any signature morphism (u, v), we associate:
- A *translation* map between the sentences of $(Term, \Pi)$ and $(Term', \Pi')$, that replaces the terms and relation symbols with their images through u and v.
- The following *reduct functor* between the categories of models of $(Term', \Pi')$

$$\frac{\cdot}{\Gamma \rhd \Delta}\ (\mathsf{Ax})\ [\Gamma \cap \Delta \neq \emptyset]$$

$$\frac{\Gamma \rhd \Delta, \varphi \quad \Gamma, \psi \rhd \Delta}{\Gamma, \varphi \Rightarrow \psi \rhd \Delta}(\mathsf{Left}\Rightarrow) \qquad \frac{\Gamma, \varphi \rhd \Delta, \psi}{\Gamma \rhd \Delta, \varphi \Rightarrow \psi}(\mathsf{Right}\Rightarrow)$$

$$\frac{\Gamma, \varphi, \psi \rhd \Delta}{\Gamma, \varphi \wedge \psi \rhd \Delta}(\mathsf{Left}\wedge) \qquad \frac{\Gamma \rhd \Delta, \varphi \quad \Gamma \rhd \Delta, \psi}{\Gamma \rhd \Delta, \varphi \wedge \psi}(\mathsf{Right}\wedge)$$

$$\frac{\Gamma, \forall x.\varphi, \varphi[T/x] \rhd \Delta}{\Gamma, \forall x.\varphi \rhd \Delta}(\mathsf{Left}\forall) \qquad \frac{\Gamma \rhd \Delta, \varphi[y/x]}{\Gamma \rhd \Delta, \forall x.\varphi}\ (\mathsf{Right}\forall)\ [y \text{ fresh}]$$

Fig. 1. Gentzen System $\mathcal{G}$

and $(Term, \Pi)$: On objects, it maps any $(Term', \Pi')$-model $\mathcal{A}' = (A', \dot{\neg}^{A'}, (A'_{(n,\pi')})_{n\in\mathbb{N},\pi'\in\Pi'_n})$ to the model $\mathcal{A} = (A, \dot{\neg}^{A}, (A_{(n,\pi)})_{n\in\mathbb{N},\pi\in\Pi_n})$ where $A = A'$, $A_{(n,\pi)} = A'_{(n,v_n(\pi))}$ and $\dot{\neg}^{A} = \rho \mapsto \overline{\rho}^{A'} \circ u$. On morphisms, it maps a function representing a model homomorphism to the same function regarded as an homomorphism between the reduct models. (Details and pictures in [23], Sec. A.4.)

Theorem 1. *TGL as organized above forms an institution that extends conservatively the institution of FOL.*

TGL Gentzen System and Completeness. The axiomatic properties of the generic notions of free variable and substitution in TGL provide enough infrastructure to obtain generic versions of classic FOL results. We are interested in a completeness theorem here (but other model-theoretic results could be also generalized). We shall use a generalization of the cut-free system in [10].

We fix a term-generic language $(Term, \Pi)$. A *sequent* is a pair written $\Gamma \rhd \Delta$, with *antecedent* Γ and *succedent* Δ (at most) countable sets of formulae, assumed to have *finite support*, in that $FV(\Gamma)$ and $FV(\Delta)$ are finite, where $FV(\Gamma) = \bigcup_{\varphi\in\Gamma} FV(\varphi)$ (and likewise for Δ). (In standard Gentzen systems for FOL, Γ and Δ are typically assumed finite, which of course implies finite support.) The sequent $\Gamma \rhd \Delta$ is called *tautological*, written $\models \Gamma \rhd \Delta$, if $\bigcap_{\varphi\in\Gamma} A_\varphi \subseteq \bigcup_{\psi\in\Delta} A_\psi$ for all models A; it is called *E-tautological* (where E is a set of sentences), written $E \models (\Gamma \rhd \Delta)$, if $A \models E$ implies $\bigcap_{\varphi\in\Gamma} A_\varphi \subseteq \bigcup_{\psi\in\Delta} A_\psi$ for all models A. If $\Gamma = \emptyset$, we write $E \models \Delta$ instead of $E \models (\Gamma \rhd \Delta)$.

We consider the Gentzen system, say $\mathcal{G}$, given by the rule schemes in Figure 1, meant to deduce TGL tautological sequents (we write Γ, φ instead of $\Gamma \cup \{\varphi\}$). Note that these rules make 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. We write $\vdash_{\mathcal{G}} \Gamma \rhd \Delta$ to mean that $\Gamma \rhd \Delta$ is deducible in $\mathcal{G}$. Similar notation will be used for the other proof systems hereafter.

Theorem 2. *$\mathcal{G}$ is sound and complete for TGL.*

Intuitionistic Term-Generic Logic (ITGL). It has the same syntax as TGL, and its Gentzen system $\mathcal{GI}$ is obtained by modifying $\mathcal{G}$ so that the succedents in sequents are no longer sets of formulae, but single formulae, as follows: **(1)** Δ is deleted from all the Right rules and is replaced by a single formula χ in (Ax) and in all the Left rules except for (Left$\Rightarrow$). **(2)** The rule (Left$\Rightarrow$) is replaced by $\dfrac{\Gamma \rhd \varphi \quad \Gamma, \psi \rhd \chi}{\Gamma, \varphi \Rightarrow \psi \rhd \chi}$ (Section A.5 in [23] gives more details.)

3 The Horn^2 Fragment of Term-Generic Logic

We next consider a fragment of TGL, called HORN^2 because it only allows formulae which are universally quantified implications whose conditions are themselves universally quantified implications of atomic formulae. A whole plethora of λ-calculi can be specified by HORN^2 formulae (see Section 4 and [23], Section C). As shown in the sequel, we can associate uniformly to these specifications complete intuitionistic proof systems that turn out to *coincide* with the originals.

In what follows, $\overline{x}$ denotes variable tuples $(x_1, \ldots, x_n)$, $\overline{T}$ term tuples $(T_1, \ldots, T_n)$, and, for a formula φ, $\varphi(\overline{x})$ indicates that φ has all its free variables *among* $\{x_1, \ldots, x_n\}$, with $\varphi(\overline{T})$ then denoting $\varphi[T_1/x_1, \ldots, T_n/x_n]$. Since variables are particular terms, given $\overline{y} = (y_1, \ldots, y_n)$, we may use the notation $\varphi(\overline{y})$ with two different meanings, with disambiguation coming from 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 $\varphi(\overline{x})$ by substituting the variables $\overline{x}$ with $\overline{y}$. (Thus, e.g., in the property $(*)$ below, $a_i(\overline{x}, \overline{y})$ is the same as a_i, where in addition we have indicated that the free variables of a_i are among $\mathit{Vars}(\overline{x}, \overline{y})$ (where $(\overline{x}, \overline{y})$ is the concatenation of the tuples $\overline{x}$ and $\overline{y}$). Later, given the tuples $\overline{z}$ and $\overline{T}$ of appropriate lengths, $a_i(\overline{z}, \overline{T})$ denotes the formula obtained from a_i by substituting the variables of $\overline{x}$ correspondingly with those of $\overline{z}$ and the variables of $\overline{y}$ correspondingly with the terms of $\overline{T}$.) For convenience, we assume the logic also contains $\top$ (meaning "true") as an atomic formula.

Let HORN^2 be the TGL fragment given by the sentences:

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

where a_i, b_i, c are atomic formulae and we assume $\mathit{Vars}(\overline{x}) \cap \mathit{Vars}(\overline{y}) = \emptyset$. We call these HORN^2 *sentences* (sometimes we shall refer to them as HORN^2 *formulae*, not forgetting though that they have no free variables). When one of the above a_i's 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 when all the a_i's are $\top$ we call the sentence $(*)$ *extensional* [25]; if, in addition, $\overline{x}$ has length 0, we obtain a Horn sentence (that is, a universal closure of a Horn formula). When all b_i's are $\top$ or $n = 0$, the whole sentence $(*)$ becomes $\forall \overline{y}.\, c(\overline{y})$. A theory E is called HORN^2, extensional, or Horn if it consists of such sentences.

Fix a term-generic language and a HORN^2 theory (i.e., specification) E over this language. In what follows, we focus on *Horn sequents*, i.e., sequents $\Gamma \triangleright d$ with Γ finite set of atomic formulae and d atomic formula, which can be deduced from E. Only Horn consequences are usually relevant for λ-calculi, and, moreover, the other more syntactically complicated consequences can be deduced from these. We let $\mathcal{K}_E$ denote the following Gentzen system for Horn sequents:

$\dfrac{\cdot}{\Gamma \triangleright d}$ (Axiom) $[d \in \Gamma]$	$\dfrac{\Gamma, a_i(\overline{z}, \overline{T}) \triangleright b_i(\overline{z}, \overline{T}) \text{ for } i = \overline{1, n}}{\Gamma \triangleright c(\overline{T})}$ (Inst-e)

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), $\overline{T}$ is a tuple of terms with the same length as $\overline{y}$, and $\overline{z}$ is a *fresh* tuple of variables with the same length as $\overline{x}$ (where "fresh" (without further attributes) means, as usual,

"fresh for everything in that context", namely: for Γ, the a_i's, the b_i's, c and $\overline{T}$). Thus (Inst-e) is a rule (more precisely, a rule-scheme) parameterized not only by e, but also by $\overline{z}$ and $\overline{T}$ as above (and by Γ, too). (More details in [23], Sec. B.)

A first result is that $\mathcal{K}_E$ deduces all intuitionistic Horn consequences of E:

Theorem 3. *$\vdash_{\mathcal{GI}} (E \cup \Gamma) \triangleright d$ iff $\vdash_{\mathcal{K}_E} \Gamma \triangleright d$ for all Horn sequents $\Gamma \triangleright d$.*

Now consider the following family of rules (Drop) = (Drop-(e,a))$_{e,a}$, parameterized by formulae $e \in E$ of the form $(*)$ and by atomic formulae a such that a is one of the a_i's (for some $i \in \{1,\ldots,n\}$):

$$\frac{\Gamma, a(\overline{z},\overline{T}) \triangleright d}{\Gamma \triangleright d}\ \text{(Drop-}(e,a)\text{)}$$

($\overline{T}$ is a tuple of terms of the same length as $\overline{y}$ and $\overline{z}$ a tuple of variables of the same length as $\overline{x}$ *fresh for* d (where $\overline{y}$ and $\overline{x}$ are the ones referred in $(*)$).)

From the point of view of forward proofs, (Drop) effectively drops $a(\overline{z},\overline{T})$. More interesting than the actual usage of (Drop) is its admissibility in a system. In a specification of a type system for a λ-calculus, $a(\overline{z},\overline{T})$ will typically have the form $z{:}T$, and closure of the system under $a(\overline{z},\overline{T})$ will be a *condensing lemma* [3]: the assumption $z{:}T$ is useless provided z is not in the succedent.

Next are our main results of this section, exploring closure under (Drop). The first gives a sufficient criterion ensuring completeness of $\mathcal{K}_E$ w.r.t. TGL models. The second gives a stronger fully syntactic criterion.

Theorem 4. *Assume that:*
-(i) If a_i is not $\top$, then $Vars(\overline{x}) \cap FV(a_i) \neq \emptyset$, for all formulae $e \in E$ of the form $(*)$ and all $i \in \overline{1,n}$.
-(ii) (Drop) is admissible in $\mathcal{K}_E$.
Then $\vdash_{\mathcal{K}_E} \Gamma \triangleright d$ iff $E \models (\Gamma \triangleright d)$ for all Horn sequents $\Gamma \triangleright d$.

Theorem 5. *Assume that all e in E of the form $(*)$ satisfy the following for all $i \in \overline{1,n}$:*
-(i) If a_i is not $\top$, then $Vars(\overline{x}) \cap FV(a_i) \neq \emptyset$.
-(ii) $Vars(\overline{y}) \cap FV(b_i) \subseteq FV(c)$.
Then (Drop) is admissible in $\mathcal{K}_E$, hence the conclusion of Theorem 4 holds.

Definition 3. *We call a* HORN^2 *theory E:*
- amenable*, if it satisfies the hypotheses of Theorem 4;*
- syntax-amenable*, if it satisfies hypothesis (i) of Theorem 4 (same as hypothesis (i) of Theorem 5);*
- strongly syntax-amenable*, if it satisfies the hypotheses of Theorem 5.*

(Thus strong syntax-amenability implies amenability.)

If E is a Horn theory, Theorems 4, 5 yield the completeness result for a well-known Hilbert system of Horn logic. More generally, amenability, hence completeness, holds trivially for extensional theories, since they have no (Drop) rules.

Thus classic TGL has, with respect to amenable theories and Horn consequences, *the same deductive power as intuitionistic TGL*. This fact will prove useful for adequacy results and completeness of the TGL models for various calculi. Because these calculi are traditionally specified following an intuitionistic pattern, an amenable HORN^2 specification E of a calculus will recover, in the system $\mathcal{K}_E$, *the represented calculus itself* – we discuss this phenomenon next.

4 Specifying calculi in Term-Generic Logic

This section illustrates the TGL λ-calculi specification style. Our running example is the typing system and reduction of *System F*, an impredicative polymorphic typed λ-calculus introduced independently in [11] and [24]. (Many other examples can be found in [23], Section C.) Its syntax modulo α-equivalence clearly forms a two-sorted TGL term syntax. The sorts are *type* and *data*, and we write $TVar$ for Var_{type} (ranged over by t, t') and $DVar$ for Var_{data} (ranged over by x, y), as well as $TTerm$ for $Term_{type}$ (ranged over by T, T') and $DTerm$ for $Term_{data}$ (ranged over by X, Y). Here is the grammar for (the raw terms out of which, by factoring to standard α-equivalence, one obtains) the terms:

$$T ::= t \mid T \to T' \mid \Pi\, t.\, T$$
$$X ::= x \mid \lambda\, x : T.\, X \mid X\, Y \mid \lambda\, t.\, X \mid X\, T$$

A *typing context* Γ is a finite set $\{x_1 : T_1, \ldots, x_n : T_n\}$ (written $x_1 : T_1, \ldots, x_n : T_n$ for brevity), where the x_i's are data variables, the T_i's are type terms, and no data variable appears twice. The typing system for System F, denoted TSF, deriving sequents $\Gamma \triangleright X : T$, is the following:

$$\frac{\cdot}{\Gamma \triangleright x : T}\ [(x:T) \in \Gamma]\quad \textsf{(SF-InVar)}$$

$$\frac{\Gamma, x : T \triangleright X : T'}{\Gamma \triangleright (\lambda x : T.X) : T \to T'}\ [x \text{ fresh for } \Gamma]\quad \textsf{(SF-Abs)}$$

$$\frac{\Gamma \triangleright X : T \to T' \quad \Gamma \triangleright Y : T}{\Gamma \triangleright X\, Y : T'}\ \textsf{(SF-App)}$$

$$\frac{\Gamma \triangleright X : T}{\Gamma \triangleright (\lambda t.X) : \Pi t.T}\ [t \text{ fresh for } \Gamma]\quad \textsf{(SF-T-Abs)}$$

$$\frac{\Gamma \triangleright X : \Pi t.T}{\Gamma \triangleright X\, T' : T[T'/t]}\ \textsf{(SF-T-App)}$$

We specify TSF as a HORN^2 theory by identifying the implicit universal quantifications and implications involved in the original system. For example, we read (SF-Abs) as: if one can type X to T' *uniformly on* x assuming x has type T, i.e., *for all* x of type T, then $\lambda x : T.\, X$ receives type $T \to T'$. But this is HORN^2! (T and T' above are not involved in any bindings relevant here, hence we can use TGL variables instead.) Below is the whole theory, $\mathcal{TSF}$, in a term-generic language over the indicated term syntax and having the infixed relation symbol ":" with arity *data* $\times$ *type*. (The colons denoting this relation, although related with, should not be confounded with the colons used as part of the term syntax – our poly-semantic usage of ":" mirrors the usage in the original system TSF.)

$(\forall x.\, x : t \Rightarrow X : t')$ $\Rightarrow (\lambda x : t.X) : t \to t'$	(Abs)
$x : t \to t' \wedge y : t \Rightarrow (x\, y) : t'$	(App)

$(\forall t.\, X : T)$ $\Rightarrow (\lambda t.X) : (\Pi t.\, T)$	(T-Abs)
$x : (\Pi t.\, T) \Rightarrow (x\, t) : T$	(T-App)

(Abs), (T-Abs) and (T-App) are axiom-schemes, parameterized by arbitrary terms X, T. In (Abs), a presumptive occurrence of x in the leftmost X is in the scope of the universal quantifier, and in the rightmost X in the scope of the λ-abstraction; similarly for t versus X and t versus T in (T-Abs). This migration of the variables x and t between scopes may look surprising at first – note however that the same situation appears in the corresponding rules ((SF-Abs) and (SF-T-Abs)) from the familiar system TSF. Thus, in (SF-Abs), any occurrence of x in the term X from the succedent of the conclusion sequent $\Gamma \triangleright (\lambda x : T.\, X) : T \to T'$

is in the scope of the λ-abstraction, while *the same* occurrence of x in X when part of the antecedent of the hypothesis sequent $\Gamma, x : T \triangleright X : T'$ is not in the scope of any binder (more precisely, is in the scope of the implicit outer binder of the sequent).

Both in the original system and in our HORN^2 specification, the assumption that T, X, etc. are terms *modulo α-equivalence* is consistent with their usage in combination with binding constructs, since, for example, the syntactic operator $(\lambda\_:\_.\_) : DVar \times TTerm \times DTerm \to DTerm$ is well defined on α-equivalence classes. Note that a concrete HORN^2 specification cannot be stated solely in terms of the logic's constructs (as is the case of representations in a *fixed* logic, like HOL) simply because TGL does not specify the term syntax, but *assumes it*. Consequently, our examples of specifications employ, at the meta-level, constructs like the above $(\lambda\_:\_.\_)$, not "purely TGL". (This paper does *not* discuss how to define and represent term syntaxes conveniently, but how to represent the structure of a calculus *on top* of a given term syntax – see also Section 5.)

One should think of the above HORN^2 axioms *semantically*, as referring to items called *data* and *types* that inhabit TGL models – hence our terminology, which distinguishes between *data* terms and variables on the one hand and *type* terms and variables on the other (compare this with the more standard terminology distinguishing between *terms* and *types* from purely syntactic presentations of λ-calculi). As usual, focussing on the semantics allows one to state the desired properties without worrying about syntactic details such as typing contexts and side-conditions; all such lower-level details can nevertheless become available when one "descends" into the deductive system of TGL.

What is the formal relationship between the original typing system TSF and the HORN^2 theory $\mathcal{TSF}$? TSF is precisely $\mathcal{K}_{\mathcal{TSF}}$ from Section 3, the Gentzen system associated to a HORN^2 theory in a uniform way. (Namely, referring to the notations of Section 3: (SF-InVar) is (Axiom), (SF-Abs) is (Inst-Abs), (SF-T-Abs) is (Inst-T-Abs), (SF-App) is (Inst-App), and (SF-T-App) is (Inst-T-App)) Therefore, not only that $\mathcal{TSF}$ *specifies* TSF, but also TSF *implements* $\mathcal{TSF}$ as its specialized deductive system. Consequently, the following adequacy result w.r.t. intuitionistic TGL is *built in* the representation (via Theorem 3):

Proposition 1. *Let $x_1, \ldots, x_n$ be distinct data variables, X data term and $T, T_1, \ldots, T_n$ type terms. Then the following are equivalent:*
(a) $\vdash_{TSF} x_1 : T_1, \ldots, x_n : T_n \triangleright X : T$.
(b) $\vdash_{\mathcal{K}_{\mathcal{TSF}}} x_1 : T_1, \ldots, x_n : T_n \triangleright X : T$.
(c) $\vdash_{\mathcal{GI}} \mathcal{TSF}, x_1 : T_1, \ldots, x_n : T_n \triangleright X : T$
(where $\mathcal{TSF}, x_1 : T_1, \ldots, x_n : T_n$ is a notation for $\mathcal{TSF} \cup \{x_1 : T_1, \ldots, x_n : T_n\}$).

In order to obtain adequacy w.r.t. classic TGL as well, we further need to notice:

Lemma 1. *$\mathcal{TSF}$ satisfies (a many-sorted version of) strong syntax-amenability.*

and then invoke Theorem 5, obtaining:

Proposition 2. *Let $x_1, \ldots, x_n$ be distinct data variables, X data term and $T, T_1, \ldots, T_n$ type terms. Then the following are equivalent:*
(a) $\vdash_{TSF} x_1 : T_1, \ldots, x_n : T_n \triangleright X : T$.
(b) $\mathcal{TSF} \models (x_1 : T_1, \ldots, x_n : T_n \triangleright X : T)$.

Next, we consider the following standard Hilbert system for reduction in System F [11, 24] (obtained from the one for the untyped λ-calculus [4] by ignoring the type annotations), denoted RSF:

$$\frac{}{(\lambda x{:}T.\,Y)X \rightsquigarrow Y[X/x]}\,(\mathsf{SF}\text{-}\beta) \qquad \frac{}{(\lambda t.\,Y)T \rightsquigarrow Y[T/t]}\,(\mathsf{SF\text{-}T}\text{-}\beta)$$

$$\frac{X \rightsquigarrow X'}{\lambda x{:}T.\,X \rightsquigarrow \lambda x{:}T.\,X'}\,(\mathsf{SF}\text{-}\xi) \qquad \frac{X \rightsquigarrow X'}{\lambda t.\,X \rightsquigarrow \lambda t.\,X'}\,(\mathsf{SF\text{-}T}\text{-}\xi)$$

$$\frac{X \rightsquigarrow X'}{X\,Y \rightsquigarrow X'\,Y}\,(\mathsf{SF\text{-}AppL}) \qquad \frac{Y \rightsquigarrow Y'}{X\,Y \rightsquigarrow X\,Y'}\,(\mathsf{SF\text{-}AppR}) \qquad \frac{X \rightsquigarrow X'}{X\,T \rightsquigarrow X'\,T}\,(\mathsf{SF\text{-}T\text{-}App})$$

Our HORN^2 specification, denoted $\mathcal{RSF}$, uses relation $\rightsquigarrow$ of arity $data \times data$.

$(\lambda x : t.\,Y)x \rightsquigarrow Y$	(β)
$(\lambda t.\,Y)t \rightsquigarrow Y$	(T-β)
$(\forall x.\,X \rightsquigarrow X') \Rightarrow \lambda x{:}t.\,X \rightsquigarrow \lambda x{:}t.\,X'$	(ξ)
$(\forall t.\,X \rightsquigarrow X') \Rightarrow \lambda t.\,X \rightsquigarrow \lambda t.\,X'$	(T-ξ)

$x \rightsquigarrow x' \Rightarrow x\,y \rightsquigarrow x'\,y$	(AppL)
$y \rightsquigarrow y' \Rightarrow x\,y \rightsquigarrow x\,y'$	(AppR)
$x \rightsquigarrow x' \Rightarrow x\,t \rightsquigarrow x'\,t$	(T-App)

Particularly interesting are our axioms for β-reduction. In (β), we employ the same variable x to indicate both the *formal parameter* of the functional expression $\lambda x : t.\,Y$ and its *actual parameter* (the occurrence of x on the right of the application from the left side of $\rightsquigarrow$). Indeed, in the latter case, as well as in any presumptive occurrences in the rightmost Y, x is exposed to the environment, hence denotes an (arbitrary) actual value in a model.

Again, $\mathcal{K}_{\mathcal{RSF}}$ is the same as RSF (modulo a standard identification of Hilbert systems with simple Gentzen systems where antecedents remain unchanged). Moreover, $\mathcal{RSF}$ is an extensional theory, hence trivially amenable, hence both intuitionistically and classically adequate:

Proposition 3. *Let X and Y be data terms. Then the following are equivalent:*
(a) $\vdash_{RSF} X \rightsquigarrow Y$.
(b) $\vdash_{\mathcal{GI}} \mathcal{RSF} \triangleright X \rightsquigarrow Y$.
(c) $\mathcal{RSF} \models X \rightsquigarrow Y$.

One can readily see that, since the relation symbols of $\mathcal{STF}$ and $\mathcal{RSF}$ are distinct, putting these theories together preserves adequacy – in other words, Propositions 1 and 2 remain true after replacing $\mathcal{SFT}$ with $\mathcal{SFT} \cup \mathcal{RSF}$ and Proposition 3 remains true after replacing $\mathcal{RSF}$ with $\mathcal{SFT} \cup \mathcal{RSF}$. In the union language, we can express relevant properties such as *type preservation*: $\forall x, y, t.\,x : t \wedge x \rightsquigarrow y \Rightarrow y : t$. The proof of such properties requires reasoning *about the calculus*, hence transcends the realm of adequate representations. To handle them, TGL needs to be extended with inductive proof schemes, such as:

$$\frac{\varphi(x) \wedge \varphi(y) \Rightarrow \varphi(x\,y) \quad ((\forall x.\,\varphi(X)) \Rightarrow \varphi(\lambda x{:}t.\,X))_{X \in DTerm}}{\forall x.\,\varphi(x)}\,(\mathsf{Induct}_{data})$$

The problem of meta-reasoning in a framework where object-level calculi are represented without explicitly encoding free variables and substitution (currently still open in frameworks such as HOAS) is not addressed in this paper, but is left as important future work.

Intuitionistic TGL adequacy (Proposition 1) holds immediately (for the same reason as for System F) for all calculi specified in [23], Section C. Classic TGL

adequacy, on the other hand, while trivial for System F (in the context of our a priori proof theory), is not so in other calculi, where strong syntax-amenability does not hold, but only syntax amenability does, and closure under (Drop), while intuitive, is not obvious to prove. Fortunately however, for most of these calculi this property coincides with a known result called the *condensing lemma* (see [3]): in a typing context $\Gamma \triangleright U : V$, an assumption $x : T$ from Γ with x fresh for U and V may be dropped without losing provability. Note that, via the propositions-as-types correspondence, representing adequately type systems in TGL also implies representing adequately proof systems for structural logics.

Sometimes a calculus does not come with a reduction relation, but with an equational theory. (Notably, a standard formulation of untyped λ-calculus [4] is equational.) For these situations, a version of TGL with equality seems a more elegant choice, but adequacy proofs along our lines seem to require more effort, since the TGL equality axioms raise problems regarding amenability (not to mention that type preservation needs to be proved beforehand for the calculus). Alternatively, one may provide semantic proofs for adequacy, taking advantage of the equivalence between the TGL models and some ad hoc models for which the calculus is known to be complete (see Section E in [23] for this approach).

5 Concluding Remarks

Summing up the contribution of this paper:
(1) We showed that the development of first-order logic is largely orthogonal to the particular syntax of terms by defining a logic, TGL, that considers terms as "black-boxes" exporting substitution and free variables and requires models to represent terms consistently. TGL forms an institution, hence allows in principle for well-structured logical specifications.
(2) TGL provides a convenient notation and intuition for defining λ-calculi, that encourages a semantic specification style. We developed some proof theory to support this specification style. Intuitionistic TGL allows immediately adequate specifications, while for classic TGL adequacy, if provable, endows the specified calculus with a *default complete semantics*.

The idea of developing first-order logic on top of an abstract term syntax, as well as our proof-theoretic results that prepare the logic in advance for adequate representations of λ-calculi, seem new.[4] We separate the discussion of related work into two (non-disjoint) topics.

One concerns semantics. The semantics that TGL offers to the specified calculi for free 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. The monographs [4, 14, 20] contain extensive presentations of these and many other loose semantics for various calculi. For a particular calculus defined as a TGL theory, the

[4] But see below the related work on HOAS and categorical models of syntax.

attached TGL semantics has all the advantages, but, naturally, also all the drawbacks, of loose semantics. It was 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 TGL semantics though is its uniform, *calculus-independent* nature. (We argue in [23] (Section E), with untyped λ-calculus and System F as witnesses, that the "general-purpose" TGL semantics of a calculus tends to be equivalent to the set-theoretic "domain-specific" one whose completeness theorem is typically worked out separately with substantial mathematical effort in the literature.)

The other topic concerns existing specification frameworks in the literature:

- Purely first-order encodings, such as combinatory logic [4], de Bruijn-style representations [6], and the calculus with explicit substitution [1]. Part of the motivation of TGL was to avoid the degree of awkwardness and auxiliary proof or execution overhead of such encodings.

- Higher-order abstract syntax (HOAS) [21, 15, 17]. This approach encodes (in a binding-preserving fashion) object-level terms into terms of a *fixed meta logic* (usually HOL or another type theory) – consequently, the interpretation of the object syntax into presumptive models of the meta logic would be indirect, filtered through the encoding. To the contrary, TGL is a *parameterized logic*, and gets *instantiated* to various calculi by importing the original term syntax *as is* and relating models to this syntax *directly* through valuations. Moreover, usually model-theoretic considerations are not the concern of HOAS, which aims at proof-theoretic adequacy alone, a property that so far seemed to require an intuitionistic meta logic; here we also developed for TGL a technique for establishing adequacy within a classic logic.

Yet, TGL representations have important similarities with HOAS encodings in variants of HOL (in the style of, e.g., [17]). For instance, our axiom-scheme (Abs) from the HORN^2 theory $\mathcal{TSF}$ may be in such an encoding $\forall X.\,(\forall x.\,x : t \Rightarrow X(x) : t') \Rightarrow Lam(X) : (t \to t')$, where $X : data \longrightarrow data$ is a second-order variable and $Lam : (data \longrightarrow data) \longrightarrow data$ is a third-order constant. A HOAS encoding has typically two parts, each requiring its own adequacy result: one deals with representing the syntax of terms, and one with representing the deductive mechanism. Because TGL does not provide a representation of syntax (but assumes one already), some of our axioms, namely those changing variable scopes, such as (Abs), are (still) axiom-schemes, just like the rules of the original calculus are rule-schemes; to the contrary, the above HOAS axiom would be a single statement. On the other hand, for the same reason (of not dealing with term syntax representation), we were able to discuss the second part, of representing the deductive mechanism, *generically, for any term syntax*, and have created a theoretical framework where adequacy for the deductive mechanisms requires minimal proof effort. "Pasting" various solutions offered by HOAS to representing terms into the TGL framework for representing deduction could allow a HOAS setting to benefit from our theorems in Section 3, as well as allow a HOAS representation of an effective-syntax fragment of TGL to bypass the need of axiom-schemes in specifications.

Categorical models of syntax in the style of [9, 16] also fall within HOAS. Typing contexts are explicitly modeled as possible worlds, types becoming presheaves. The presheaf structure of λ-terms from [16] and the substitution algebras from [9] are roughly equivalent to our term syntaxes (whose presheaf structure would come, just like in the concrete cases, from classifying terms by their sets of free variables). The model theory of the these categorical settings follows a different approach than ours though – they require the models to support *substitution within themselves and between each other* (hence to be inhabited by syntactic items such as (abstract) terms and variables), while we require the models to allow *valuations from a fixed term model*.

- Nominal logic (NL) [22]. It stands somewhere in between purely first-order encodings and HOAS, as it captures object-level bindings, but not substitution, by corresponding meta-level mechanisms. The NL terms with bindings form term syntaxes in our sense. Like in the categorical approaches mentioned above and unlike TGL models, NL models are inhabited by abstract syntactic objects (having, e.g., free names that can be swapped/permuted) rather than constituting "pure" FOL-like semantics.

- Explicitly closed families of functionals (ECFFs) [2] (a.k.a. binding algebras [27]). In the tradition of HOL a la Church, all bindings are reduced there to functional abstraction. Their terms form term syntaxes in our sense, and ECFFs are particular cases of TGL models.

- Binding logic (BL) [8]. It is a first-order logic defined on top of a general notion of syntax with binding, allowing bindings in both operations and atomic predicates. BL models reflect the bindings *functionally* (similarly to [2], [27]). While BL terms form TGL term syntaxes, it appears that the class of BL models is strictly embedded in that of TGL models for TGL terms syntax instantiated to a BL language of terms.

- Hereditary Harrop Formulae (HHF). For the FOL and HOL instances of TGL, HORN^2 formulae are particular cases of such formulae, advocated in [19] for logic programming. Our proof-theoretic results from Section 3 seem HORN^2-specific, a generalization to HHF not being apparent.

- In the general realm of logical and algebraic specifications, a salient framework is that of institutions [12, 7]. Like TGL, the notion of institution does not represent logical systems by *encoding* them, but by becoming *instantiated* to them. Since we showed that TGL is itself an institution,[5] our work in this paper offers to the λ-calculi adequately specifiable in TGL institutional citizenship, hence the algebraic arsenal of tools and techniques from institution theory [26].

References

1. M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *J. Funct. Program.*, 1(4):375–416, 1991.
2. P. Aczel. Frege structures and the notions of proposition, truth and set. In *The Kleene Symposium*, pages 31–59. North Holland, 1980.

[5] In fact, we showed that TGL is an institution endowed with a (sound and complete) proof system, making it a *logical system* in the sense of [18].

3. H. Barendregt. Introduction to generalized type systems. *J. Funct. Program.*, 1(2):125–154, 1991.
4. H. P. Barendregt. *The Lambda Calculus*. North-Holland, 1984.
5. G. Birkhoff. On the structure of abstract algebras. *Proceedings of the Cambridge Philosophical Society*, 31:433–454, 1935.
6. N. Bruijn. λ-calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indag. Math*, 34(5):381–392, 1972.
7. R. Diaconescu. *Institution-independent Model Theory*. Birkhauser, 2008.
8. G. Dowek, T. Hardin, and C. Kirchner. Binding logic: Proofs and models. In *LPAR*, volume 2514 of *Lecture Notes in Computer Science*, pages 130–144, 2002.
9. M. Fiore, G. Plotkin, and D. Turi. Abstract syntax and variable binding. In *Proc. 14$^{\text{th}}$ LICS Conf.*, pages 193–202. IEEE, 1999.
10. J. H. Gallier. *Logic for computer science. Foundations of automatic theorem proving*. Harper & Row, 1986.
11. 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. In J. Fenstad, editor, *2nd Scandinavian Logic Symposium*, pages 63–92. North Holland, 1971.
12. J. Goguen and R. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, January 1992.
13. J. Goguen and J. Meseguer. Order-sorted algebra I. *Theoretical Computer Science*, 105(2):217–273, 1992.
14. C. A. Gunter. *Semantics of Programming Languages*. MIT Press, 1992.
15. R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. In *Proc. 2$^{\text{nd}}$ LICS Conf.*, pages 194–204. IEEE, 1987.
16. M. Hofmann. Semantical analysis of higher-order abstract syntax. In *Proc. 14$^{\text{th}}$ LICS Conf.*, pages 204–213. IEEE, 1999.
17. R. C. McDowell and D. A. Miller. Reasoning with higher-order abstract syntax in a logical framework. *ACM Trans. Comput. Logic*, 3(1):80–136, 2002.
18. J. Meseguer. General logics. In H.-D. Ebbinghaus et al., editors, *Proceedings, Logic Colloquium 1987*, pages 275–329. North-Holland, 1989.
19. D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Ann. Pure Appl. Logic*, 51(1-2):125–157, 1991.
20. J. C. Mitchell. *Foundations for Programming Languages*. MIT Press, 1996.
21. F. Pfenning and C. Elliot. Higher-order abstract syntax. In *PLDI '88*, pages 199–208. ACM Press, 1988.
22. 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.
23. A. Popescu and G. Roşu. Term-generic logic. Tech. Rep. Univ. of Illinois at Urbana-Champaign UIUCDCS-R-2009-3027, 2009.
24. J. C. Reynolds. Towards a theory of type structure. volume 19 of *Lecture Notes in Computer Science*, pages 408–423. Springer-Verlag, 1974.
25. G. Roşu. Extensional theories and rewriting. In *ICALP'04*, volume 3142 of *Lecture Notes in Computer Science*, pages 1066–1079, 2004.
26. D. Sannella and A. Tarlecki. *Foundations of Algebraic Specifications and Formal Program Development*. Cambridge University Press. To appear. (Ask authors for current version at tarlecki@mimuw.edu.pl).
27. Y. Sun. An algebraic generalization of Frege structures - binding algebras. *Theoretical Computer Science*, 211(1-2):189–232, 1999.