A First-Order Logic with First-Class Types

Peter H. Schmitt, Mattias Ulbrich, and Michael Walter

University of Karlsruhe
Institute for Theoretical Computer Science
D-76128 Karlsruhe, Germany
{pschmitt,mulbrich}@ira.uka.de, michael.walter@gmail.com

Abstract. This paper presents a strongly complete calculus for a first-order statically-typed predicate logic with first-class types, type predicates and casts, provided that the type hierarchy is Noetherian. We show that this restriction cannot be relaxed.

1 Introduction

In traditional Mathematical Logic, typed first-order logic or, as it was mostly called, many-sorted first-order logic played only a marginal role. The fact that it could in principle be reduced to first-order logic with unary type predicates was excuse enough to pass over it. This started to change when logic turned into an intrinsic topic in Computer Science. It became apparent that the use of types could provide useful guidance for proof search in automated theorem proving. This triggered papers like [1–5]. More recently formal verification of object-oriented programs provided an additional motivation to develop typed first-order calculi. A calculus tailored towards reasoning about typed Java programs including `instanceof` predicates and `cast` functions but no generic types was developed by Martin Giese in [6] and is successfully used within the KeY system [7, 8]. A first investigation into the logical issues involved in the verification of object-oriented programs with generic types was done in [9].

The aim of this paper is to extend the logic from [6] so that the type hierarchy is part of the domain. Thus types are *first-class* values which can be subjects of formulae; in particular, they can be quantified over. Our logic has a sound and complete calculus if and only if the type hierarchy is Noetherian; completeness is to be understood as *strong completeness*, i.e., the calculus will derive all correct semantical consequences from arbitrary sets of assumptions and not only tautologies. This will follow from an axiomatization in the theory of [6] for which we will prove a similar characterization generalizing the main theorem of that paper.

Outline. This paper is structured as follows. In Subsection 2.1–2.3 we present the first-order statically-typed predicate logic **TFOL** with first-class types, type predicates and casts. In Subsection 2.4 we review the results from [6] and prove the necessary generalization. Finally Subsection 2.5 covers the main axiomatization results of this paper. Section 3 presents a way to turn this axiomatization

into a more efficient tableau calculus. Section 4 evaluates possible applications and points out future work, and we wrap up with conclusions in the last section.

Prerequisites. In the following we will make repeated use of the well-known fact that every logic with a sound and complete calculus is compact; we will refer to this fact as the *compactness theorem*.

2 A Logic with First-Class Types

2.1 Types

As the symbols in our signatures will be statically typed, we first need to clarify our notions of types and type hierarchies. The following definition is inspired by [6] and [7].

Definition 1 (Type hierarchy). *A* type hierarchy *is a set $\mathcal{T}$ (the set of* types*) together with a partial order $\sqsubseteq$ (the* subtype *relation) such that*

- *$\mathcal{T}$ is closed under (finite) greatest lower bounds, written $A \sqcap B$ (also called the* intersection type *of A and B)*
- *there is an* empty type *$\bot \in \mathcal{T}$ and a* universal type *$\top \in \mathcal{T}$ such that*

$$\bot \sqsubseteq A \sqsubseteq \top \qquad \forall A \in \mathcal{T}$$

- *there is a* type of all types *$\mathbb{T} \in \mathcal{T}$ different from $\bot$ and $\top$ such that the chain*

$$\bot \sqsubseteq \mathbb{T} \sqsubseteq \top$$

  cannot be refined

We say that A is a subtype *of B if $A \sqsubseteq B$. In that situation we also call B a* supertype *of A; this defines the supertype relation $\sqsupseteq$. Note that we do* not *require the set of types to be finite.*

We call two type hierarchies equivalent *if there is a bijection respecting the subtype relation which maps the special types to their respective counterparts.*

Remark 2 (Java type hierarchies). Given a Java program there is a natural way of defining a type hierarchy (in the sense of our definition) which preserves the class hierarchy: Take the class hierarchy, add new types $\top$ and $\bot$ and adjoin all value types as well as $\mathbb{T}$ as siblings of the class hierarchy (see the KeY book [8, pp. 24–25] for the details which are slightly more involved).

The following property will later turn out to be a sensible restriction assuring completeness of our calculus.

Definition 3 (Noetherian type hierarchy). *A type hierarchy $(\mathcal{T}, \sqsubseteq)$ is called* Noetherian *if every infinite descending chain eventually becomes stationary. That is, for every chain*

$$A_0 \sqsupseteq A_1 \sqsupseteq \dots$$

there exists $n_0 \in \mathbb{N}_0$ such that $A_n = A_{n_0}$ for all $n \geq n_0$.

Remark 4. Java generic classes (as introduced in version 5 of the programming language) inherently lead to non-Noetherian type systems. Indeed, any unbounded generic class `G<T>` leads to an infinite descending proper chain

$$\top \sqsupseteq \texttt{G<?>} \sqsupseteq \texttt{G<G<?>>} \sqsupseteq \dots$$

Even without wildcard parametrizations we still have an infinite descending proper chain

$$\texttt{Object} \sqsupseteq \texttt{Object[]} \sqsupseteq \texttt{Object[][]} \sqsupseteq \dots$$

In both cases Java Generics provide us with a way to actually make use of an unbounded number of these types at runtime (see the example of [9, pp. 82–83]). We will get back later to discussing this issue in section 4.

Let us now fix an arbitrary type hierarchy $(\mathcal{T}, \sqsubseteq)$ for the remainder of this section.

2.2 Syntax

Definition 5 (Signature). *A* signature *for our logic consists of (disjoint) sets of variable, function and predicate symbols. Variable symbols have a type, function symbols have both argument types and a result type, and predicate symbols have just argument types; we write $v : A$, $f : A_1 \times \dots \times A_n \to A$ and $p : A_1 \times \dots \times A_n$, respectively. A function symbol without any arguments is called a constant symbol.*

We require that the empty type $\bot$ does neither occur as the type of a variable symbol nor as the result type of a function symbol.

Furthermore, the signature shall contain the following reserved symbols:

- $\doteq : \top \times \top$, *the* equality *predicate symbol,*
- $\in : \top \times \mathbb{T}$, *the* type predicate *symbol,*
- $\sqsubseteq : \mathbb{T} \times \mathbb{T}$, *the* subtype predicate *symbol,*
- $\sqcap : \mathbb{T} \times \mathbb{T} \to \mathbb{T}$, *the* type intersection *function, and*
- $\text{cast}_A : \top \to A$, *the* type cast *function symbols, for every type $\bot \neq A \in \mathcal{T}$,*
- *constant symbols $A :\to \mathbb{T}$ for every type $A \in \mathcal{T}$*

In addition, there shall be an infinite number of constant and variable symbols of every type except $\bot$ (this is a usual requirement for tableau calculi to work).

Remark 6. We did not introduce a binary cast function symbol since it cannot be given a useful type without the sophisticated machinery of dependent types. Indeed, the only signature we could assign to such a function in our framework is $\text{cast} : \mathbb{T} \times \top \to \top$, but then we could not use an expression like $\text{cast}(A, x)$ in places where a term of static type A is expected.

Let us now also fix a signature for the remainder of this section.

Definition 7 (Term). *The system of sets* $\{T_A\}_{A\in\mathcal{T}\setminus\{\bot\}}$ *of* terms of static type *A is the least system of sets such that*

- $v \in T_A$ *for any variable symbol* $v : A$,
- $f(t_1, \ldots, t_n) \in T_A$ *for any function symbol* $f : A_1 \times \ldots \times A_n \to A$ *and terms* $t_i \in T_{A_i'}$ *with* $A_i' \sqsubseteq A_i$.

The static type *of a term $t \in T_A$ is designated by $\sigma(t) := A$. Note that there are no terms of static type $\bot$ (by definition of the signature).*

A subterm *of a term is inductively defined as follows: Every term is a subterm of itself, and subterms of the arguments t_i are subterms of the entire term as well.*

Definition 8 (Formula). *The set of* formulae *F is the least set such that*

- $p(t_1, \ldots, t_n) \in F$ *for any predicate symbol* $p : A_1 \times \ldots \times A_n$ *and terms* $t_i \in T_{A_i'}$ *with* $A_i' \sqsubseteq A_i$,
- $\neg\varphi$, $\varphi \vee \psi$, $\varphi \wedge \psi$, $\varphi \to \psi \in F$ *for any two formulae* $\varphi, \psi \in F$,
- $\forall v.\varphi$, $\exists v.\varphi$ *for any variable symbol* $v : A$ *and formula* $\varphi \in F$.

Notation 9. We shall use infix notation where customary (unless there is danger of confusion), e.g., we will write $t \sqsubseteq A$ instead of $\sqsubseteq(t, A)$. In addition, we will write casts using the Javaesque notation $(A)t$ instead of $\mathrm{cast}_A(t)$. Finally, when we write $\leftrightarrow$, we always mean the obvious expansion in terms of $\to$.

2.3 Semantics

Definition 10 (Structure). *A* structure $\mathcal{S} = (\mathcal{D}, \delta, \mathcal{I})$ *consists of a* domain *of values of terms, a* typing function *δ and an* interpretation *$\mathcal{I}$ of symbols in the signature.*

The typing function assigns $\delta : \mathcal{D} \to \mathcal{T}$ assigns to every domain element $x \in \mathcal{D}$ its dynamic type *$\delta(x) \in \mathcal{T}$. We define the set of domain elements of subtype of $A \in \mathcal{T}$ as*

$$\mathcal{D}_A := \{x \in \mathcal{D} : \delta(x) \sqsubseteq A\} \ .$$

An element of $\mathcal{D}_A$ is called an instance of *type A.*

We require $\mathcal{D}_\bot = \emptyset$ and $\mathcal{D}_A \neq \emptyset$ for all other types $\bot \neq A \in \mathcal{T}$. The motivation behind these requirements is twofold: First, they imply $\mathcal{D}_\mathbb{T} \cap \mathcal{D}_A = \mathcal{D}_\bot = \emptyset$ for every type $A \neq \top, \mathbb{T}$, which is sensible. Second, they guarantee that universal quantification is never trivial (there are no variables of type $\bot$); this avoids certain technicalities for the tableau calculus.

The interpretation of a function symbol $f : A_1 \times \ldots \times A_n \to A$ shall be a function

$$\mathcal{I}(f) : \mathcal{D}_{A_1} \times \ldots \times \mathcal{D}_{A_n} \to \mathcal{D}_A \ .$$

Likewise, to every predicate symbol $p : A_1 \times \ldots \times A_n$ there is assigned a relation

$$\mathcal{I}(p) \subseteq \mathcal{D}_{A_1} \times \ldots \times \mathcal{D}_{A_n} \ .$$

The predefined symbols have to satisfy the following requirements:

1. $\mathcal{I}(\doteq) = \{(x, x) : x \in \mathcal{D}\}$
2. $(\mathcal{D}_\mathbb{T}, \mathcal{I}(\sqsubseteq))$ *is a type hierarchy with empty type* $\mathcal{I}(\bot)$*, universal type* $\mathcal{I}(\top)$*, type of all types* $\mathcal{I}(\mathbb{T})$ *and intersection types given by* $\mathcal{I}(\sqcap)$
3. *The interpretation mapping* $\mathcal{I} : \mathcal{T} \to \mathcal{D}_\mathbb{T}$ *is an injection, and*

$$A \sqsubseteq B \Leftrightarrow (\mathcal{I}(A), \mathcal{I}(B)) \in \mathcal{I}(\sqsubseteq) \qquad (\forall A, B \in \mathcal{T})$$

4. *Greatest lower bounds in* $\mathcal{I}(\mathcal{T})$ *are greatest lower bounds in* $\mathcal{D}_\mathbb{T}$*, i.e.*

$$\mathcal{I}(A \sqcap B) = \mathcal{I}(\sqcap)(\mathcal{I}(A), \mathcal{I}(B)) \qquad (\forall A, B \in \mathcal{T})$$

5. *Type predicates are compatible with the structure:*

$$\mathcal{I}(\in)|_{\mathcal{D} \times \mathcal{I}(\mathcal{T})} = \bigcup_{A \in \mathcal{T}} \mathcal{D}_A \times \{\mathcal{I}(A)\}$$

6. *Type predicates respect the subtype relation:*

$$(a, b) \in \mathcal{I}(\sqsubseteq) \Rightarrow ((x, a) \in \mathcal{I}(\in) \Rightarrow (x, b) \in \mathcal{I}(\in))$$
$$(x, a), (x, b) \in \mathcal{I}(\in) \Rightarrow (x, \mathcal{I}(\sqcap)(a, b)) \in \mathcal{I}(\in)$$

   for all $x \in \mathcal{D}$ *and* $a, b \in \mathcal{D}_\mathbb{T}$
7. *Type casts are compatible with the structure:*

$$\mathcal{I}(\mathrm{cast}_A)|_{\mathcal{D}_A} = \mathrm{id}_{\mathcal{D}_A} \quad \textit{and} \quad \mathcal{I}(\mathrm{cast}_A)(\mathcal{D}) \subseteq \mathcal{D}_A$$

   for all types $\bot \neq A \in \mathcal{T}$

Remark 11. Informally stated, we require $\mathcal{D}_\mathbb{T}$ to be a type hierarchy containing $\mathcal{T}$ and that the built-ins agree and/or behave sane on $\mathcal{D}$ and $\mathcal{T}$; such structures actually exist since there are always structures with $\mathcal{D}_\mathbb{T} = \mathcal{T}$ (such that the built-ins agree with their meta-level counterparts).

On the other hand,it is *not* obvious why we did not simply require the domain of types to be equal to $\mathcal{T}$. We will see in Remark 17 that doing so would immediately imply the nonexistence of a sound and complete calculus for our logic. This is the motivation for our more "open" interpretation.

We did not assign an interpretation to variable symbols; this is taken care of by the following definition.

Definition 12 (Variable assignment). *A* variable assignment *with respect to a given structure is a map* β *assigning a domain element* $\beta(v) \in \mathcal{D}_A$ *to every variable symbol* $v : A$ *in the signature.*

We denote by β_v^x *the* modification *of* β *which assigns* $x \in \mathcal{D}_A$ *to* $v : A$*.*

Definition 13 (Value). *We inductively define the* value *of a term* t *with respect to a structure* $\mathcal{S} = (\mathcal{D}, \delta, \mathcal{I})$ *and variable assignment* β *as follows.*

$$\mathrm{val}_{\mathcal{S},\beta}(v) := \beta(v)$$
$$\mathrm{val}_{\mathcal{S},\beta}(f(t_1, \ldots, t_n)) := \mathcal{I}(f)(\mathrm{val}_{\mathcal{S},\beta}(t_1), \ldots, \mathrm{val}_{\mathcal{S},\beta}(t_n))$$

We simply write $\mathrm{val}_\mathcal{S}(t)$ *if* t *is a ground term.*

The following proposition shows that our semantics respects static typing; it is easily proved via structural induction.

Proposition 14. *Let $\mathcal{S} = (\mathcal{D}, \delta, \mathcal{I})$ be a structure, β a variable assignment and t a term. Then*

$$\delta(\mathrm{val}_{\mathcal{S},\beta}(t)) \sqsubseteq \sigma(t) \ .$$

Definition 15 (Validity). *We inductively define* validity *of a formula with respect to a structure $\mathcal{S} = (\mathcal{D}, \delta, \mathcal{I})$ and variable assignment β as follows.*

$$\begin{aligned}
&\mathcal{S}, \beta \models p(t_1, \ldots, t_n) \ :\Leftrightarrow\ (\mathrm{val}_{\mathcal{S},\beta}(t_1), \ldots, \mathrm{val}_{\mathcal{S},\beta}(t_n)) \in \mathcal{I}(p)\\
&\mathcal{S}, \beta \models \neg\varphi \ :\Leftrightarrow\ \mathcal{S}, \beta \not\models \varphi\\
&\mathcal{S}, \beta \models \varphi \vee \psi \ :\Leftrightarrow\ (\mathcal{S}, \beta \models \varphi) \text{ or } (\mathcal{S}, \beta \models \psi)\\
&\mathcal{S}, \beta \models \varphi \wedge \psi \ :\Leftrightarrow\ (\mathcal{S}, \beta \models \varphi) \text{ and } (\mathcal{S}, \beta \models \psi)\\
&\mathcal{S}, \beta \models \varphi \rightarrow \psi \ :\Leftrightarrow\ (\mathcal{S}, \beta \models \varphi) \text{ implies } (\mathcal{S}, \beta \models \psi)\\
&\mathcal{S}, \beta \models \forall v.\varphi \ :\Leftrightarrow\ \text{for all } x \in \mathcal{D}_{\sigma(v)} \text{ we have } \mathcal{S}, \beta^x_v \models \varphi\\
&\mathcal{S}, \beta \models \exists v.\varphi \ :\Leftrightarrow\ \text{there exists } x \in \mathcal{D}_{\sigma(v)} \text{ such that } \mathcal{S}, \beta^x_v \models \varphi
\end{aligned}$$

We simply write $\mathcal{S} \models \varphi$ if φ is a closed formula. In this case we also say that $\mathcal{S}$ satisfies φ.

Let $\mathcal{A}$ be a set of closed formulae. We say that φ follows from the assumptions *$\mathcal{A}$, written $\mathcal{A} \models \varphi$, if it is valid in all structures that satisfy all of the assumptions.*

Definition 16 (TFOL). *Our definitions of formulae (Def. 8), structures (Def. 10) and validity (Def. 15) define a logic which we will denote by* **TFOL**.

Remark 17. We will now make precise Remark 11. Consider the *fixed logic* $\mathbf{TFOL}^{\mathrm{fix}}$ which is defined as the modification of **TFOL** such that the domain of types is fixed to be identical to the type hierarchy. More precisely, we require that $\mathcal{I} : \mathcal{T} \to \mathcal{D}_{\mathbb{T}}$ is a *bijection*. Now fix any infinite type hierarchy $\mathcal{T}$ and signature. Let $c : \mathbb{T}$ be a constant symbol. Then

$$\Phi := \{\neg(c \doteq A) : A \in \mathcal{T}\}$$

is an unsatisfiable set of $\mathbf{TFOL}^{\mathrm{fix}}$ formulae (since the constant symbols generate the domain of type $\mathbb{T}$), but every finite subset of it is satisfiable (choose any type not excluded by the subset). In view of the compactness theorem, we have thus proved the following theorem.

Theorem 18. *For an arbitrary infinite type hierarchy there is no calculus for $\mathbf{TFOL}^{fix}$ which is both sound and complete.*

For our logic though we will be able to show that it is axiomatizable in another first-order logic (Thm. 27) which in turn can be shown to be sound and complete for a broad set of type hierarchies (Thm. 23); this will yield a sound and complete calculus for **TFOL** (Cor. 28).

2.4 Extended logic and Giese's logic

In this rather technical subsection we will extend our logic such that it contains (a slight modification of) Giese's logic from [6].

Definition 19 (Extended logic). *We define the* extended logic $\mathbf{TFOL}^\star$ *to be the logic* **TFOL** *with the following modifications: Signatures shall contain the* unary type predicate *symbols* $\in\!A : \top$ *for every type* $A \in \mathcal{T}$*, and every structure* $\mathcal{S} = (\mathcal{D}, \delta, \mathcal{I})$ *has to assign the interpretation* $\mathcal{I}(\in\!A) = \mathcal{D}_A$*.*

Remark 20. The logic $\mathbf{TFOL}^\star$ thus defined is a *definitional extension* of **TFOL**. Indeed, $\mathbf{TFOL}^\star$ type hierarchies and signatures are **TFOL** type hierarchies and signatures, respectively, and it is easy to see that a **TFOL** structure is a $\mathbf{TFOL}^\star$ structure if and only if it satisfies the following set of axioms:

$$\Psi := \{\forall v. \in\!A(v) \leftrightarrow \in(v, A) : A \in \mathcal{T}\}$$

The following lemma follows easily from this axiomatization.

Lemma 21. *There is a sound and complete calculus for* **TFOL** *if and only if there is a sound and complete calculus for* $\mathbf{TFOL}^\star$*.*

Proof. ($\Rightarrow$) Fix a $\mathbf{TFOL}^\star$ type hierarchy and signature and let φ be a single and $\mathcal{A}$ an arbitrary set of $\mathbf{TFOL}^\star$ formulae. By the preceding remark we see that $\mathcal{A} \models_{\mathbf{TFOL}^\star} \varphi$ if and only if $\Psi \cup \mathcal{A} \models_{\mathbf{TFOL}} \varphi$. But this means that, given a sound and complete calculus for **TFOL**, we can define a sound and complete calculus for $\mathbf{TFOL}^\star$ as follows:

$$\mathcal{A} \vdash_{\mathbf{TFOL}^\star} \varphi \quad :\Leftrightarrow \quad \Psi \cup \mathcal{A} \vdash_{\mathbf{TFOL}} \varphi$$

($\Leftarrow$) Since we can always rename symbols in the given **TFOL** signature such that it does not contain the unary type predicate symbols, the claim is obvious: Adjoin the unary type predicate symbols such that we also have a $\mathbf{TFOL}^\star$ signature. Then given a **TFOL** formula for the original signature, there is a unique way of extending a **TFOL** model to a $\mathbf{TFOL}^\star$ model, and conversely, every $\mathbf{TFOL}^\star$ model for φ is a **TFOL** model by restriction. Hence, we can simply use the sound and complete $\mathbf{TFOL}^\star$ calculus which is assumed to exist. □

We will now give a proper definition of our variant of Giese's first-order logic with subtyping from [6].

Definition 22 (Giese's logic). Giese's logic *which we will designate by* $\mathbf{TFOL}^0$ *is just the logic* $\mathbf{TFOL}^\star$ *with the following modifications:*

- *Type hierarchies need not contain the empty type* $\bot$ *nor the universal type* $\top$ *nor a type of all types* $\mathbb{T}$*.*
- *Signatures only need to contain the equality predicate* $\doteq$ *and the unary type predicates* $\in\!A$ *and casts* cast_A*.*
- *Interpretations are predefined only for these symbols.*

The following theorem is a generalization of the completeness theorem from [6].

Theorem 23 (Soundness and completeness for $\mathbf{TFOL}^0$). *There is a sound and complete calculus for* $\mathbf{TFOL}^0$ *and Noetherian type hierarchies.*

More precisely, if we fix a Noetherian $\mathbf{TFOL}^0$ *type hierarchy and signature we have the following: A formula* φ *follows from a set of assumptions* $\mathcal{A}$ *if and only if it can be proved using the calculus:*

$$\mathcal{A} \models_{\mathbf{TFOL}^0} \varphi \quad \Leftrightarrow \quad \mathcal{A} \vdash_{\mathbf{TFOL}^0} \varphi$$

Proof. In [6] Giese defines a first-order logic with subtyping. Let us take this logic, remove the cast to $\bot$ and require $\mathcal{D}_\bot = \emptyset$. The resulting logic (for our fixed type hierarchy and signature) is just $\mathbf{TFOL}^0$.

In that paper Giese also presents a tableau calculus and demonstrates its weak completeness (that is, for the case $\mathcal{A} = \emptyset$). The calculus is still sound for $\mathbf{TFOL}^0$ (since we have no variables of type $\bot$, we do not run into trouble by quantifying over an empty domain). Now let us add to his calculus the following rules:

$$\frac{t \sqsubseteq \bot}{\square}\ \text{close-}\bot \qquad\qquad \frac{}{\varphi}\ \text{assumption}$$

for every assumption $\varphi \in \mathcal{A}$. Soundness of these rules and hence of the resulting calculus is evident.

It remains to prove that our calculus is complete. That is, we have to show the following (recall that tableau calculi are used to prove *unsatisfiability*):

> If there is no closed tableau for $\neg\varphi$, then there is a model for $\mathcal{A} \cup \{\neg\varphi\}$, i.e. $\mathcal{A} \not\models \varphi$

We will now sketch the modifications needed to make Giese's completeness proof [6, §4, pp. 129–137] work in our extended setting (using his terminology).

(i) The assumption that "there will still always be only finitely many types in a tableau" [6, pp. 130] no longer needs to be true in our setting. But since our type hierarchy is assumed to be Noetherian, the definition of the *most specific known type* $\kappa_H(t)$ is still valid:

> Let H be a type-saturated tableau and t be a term in H. Define
>
> $$K := \{A \in \mathcal{T} : t \sqsubseteq A \in H\}$$
>
> We need to show that there is a minimum with respect to $\sqsubseteq$; it will be unique since $\sqsubseteq$ is a partial order.
> By saturation with respect to type-static, we have $\sigma(t) \in K$, hence $K \neq \emptyset$. Thus, we can choose $A_0 \in K$. If A_0 is a minimum, we are done. Otherwise there is $A_1 \in K$ such that $A_0 \not\sqsubseteq A_1$. By saturation with respect to type-$\sqcap$, we have $A_0 \sqcap A_1 \in K$. Also, $A_0 \sqcap A_1$ must be a proper subtype of A_0 (otherwise $A_0 = A_0 \sqcap A_1 \sqsubseteq A_1$, contradiction). It is clear how to continue this construction inductively.

Now, if none of the intersection types were a minimum, we would arrive at an infinite descending proper chain of types

$$A_0 \sqsupseteq A_0 \sqcap A_1 \sqsupseteq A_0 \sqcap A_1 \sqcap A_2 \sqsupseteq \dots$$

But this is a contradiction.

In particular, $t \sqsubseteq \kappa_H(t) \in H$ for every term t in a type-saturated tableau H.

(ii) We modify [6, Def. 7] such that a *saturated branch* now also has to (1) invoke the apply-$\bot$ rule whenever it is possible and (2) contain all assumptions $A \in \mathcal{A}$.

(iii) By (i) we see that a term t will still always be equipped with a superscript cast for its most specific known type $\kappa_H(t)$ (as stated in [6, p. 131]). Consequently, all proofs still work out the same.

(iv) In particular, [6, Lemma 1] together with (ii) guarantees that normalized terms on open saturated branches are never equipped with a superscript cast to $\bot$. It follows that the domain of the model $\mathcal{S}$ constructed in the proof satisfies $\mathcal{D}_\bot = \emptyset$; thus it is a structure in the sense of $\mathbf{TFOL}^0$.

(v) Finally, in the claim of the model lemma [6, Lemma 7] we can also state that $\mathcal{S}$ is a model for $\mathcal{A}$. This is evident since $\mathcal{S}$ is a model for the saturated tableau branch H, and we have $\mathcal{A} \subseteq H$ by (ii). □

Example 24. On the other hand let us consider a $\mathbf{TFOL}^0$ type hierarchy which is *not* Noetherian. Then there is an infinite descending proper chain of types

$$A_0 \sqsupseteq A_1 \sqsupseteq A_2 \sqsupseteq \dots$$

Suppose the set of common subtypes

$$\mathcal{B} := \{B \in \mathcal{T} : B \sqsubseteq A_n \ \forall n \in \mathbb{N}_0\}$$

is nonempty (in particular, this is the case if there is an empty type $\bot$). Then the infinite set of formulae $\Gamma := \Gamma_1 \cup \Gamma_2$ with

$$\Gamma_1 := \{c \sqsubseteq A_n : n \in \mathbb{N}_0\} \quad \text{and} \quad \Gamma_2 := \{\neg(c \sqsubseteq B) : B \in \mathcal{B}\}$$

(where $c : A_0$ is a constant) has no model. Indeed, suppose $\mathcal{S} = (\mathcal{D}, \delta, \mathcal{I})$ is such a model. Then in particular $\mathcal{S}$ is a model for Γ_1, thus $\delta(\mathcal{I}(c)) \in \mathcal{B}$. But this contradicts Γ_2.

On the other hand it is clear that every finite subset of Γ is satisfiable. Thus by the compactness theorem there is no a calculus which is both sound and complete.

The preceding example shows that for a broad range of type hierarchies Noetherianity also is a necessary condition. In summary:

Corollary 25 (Characterization of the existence of a sound and complete calculus for $\mathbf{TFOL}^0$). *Suppose the type hierarchy contains an empty type $\bot$. Then there is a sound and complete calculus for $\mathbf{TFOL}^0$ if and only if the type hierarchy is Noetherian.*

Remark 26. It follows using the same technique as in Lemma 21 that if we can axiomatize a logic in $\mathbf{TFOL}^0$, then this logic also has a sound and complete calculus; we use this idea in the following subsection.

But another consequence is the following: In view of Remark 17 this also means that although so far we have not ruled out the existence of a sound and weakly complete calculus for $\mathbf{TFOL}^{\text{fix}}$, there is no hope of arriving at such a calculus by axiomatization in $\mathbf{TFOL}^0$. This is another indication that the fixed semantics are too restricting.

2.5 Axiomatization

In this subsection we axiomatize the extended logic in Giese's logic. As a consequence we will be able to characterize the existence of a sound and complete calculus for **TFOL**.

Theorem 27 (Axiomatization of $\mathbf{TFOL}^\star$ in $\mathbf{TFOL}^0$). *The extended logic* $\mathbf{TFOL}^\star$ *can be axiomatized using the (possibly infinite) set* $\mathbf{TFOL}^0$ *formulae* $\Phi := \Phi_2 \cup \ldots \cup \Phi_6$ *with*

$$\begin{aligned}
\Phi_2 :=& \{\forall v.\forall w.\forall x.v \sqsubseteq v \wedge (v \sqsubseteq w \wedge w \sqsubseteq x \to v \sqsubseteq x) \wedge (v \sqsubseteq w \wedge w \sqsubseteq v \to v \doteq w)\} \\
&\cup \{\forall v.\forall w.\sqcap(v,w) \sqsubseteq v \wedge \sqcap(v,w) \sqsubseteq w \wedge (\forall x.x \sqsubseteq v \wedge x \sqsubseteq w \to x \sqsubseteq \sqcap(v,w))\} \\
&\cup \{\forall v.\bot \sqsubseteq v \wedge v \sqsubseteq \top\} \\
&\cup \{\forall v.(v \sqsubseteq \mathbb{T} \to v \doteq \bot \wedge v \doteq \mathbb{T}) \wedge (\mathbb{T} \sqsubseteq v \to v \doteq \mathbb{T} \wedge v \doteq \top)\} \\
\Phi_3 :=& \{A \sqsubseteq B : A, B \in \mathcal{T} \text{ with } A \sqsubseteq B\} \cup \{\neg(A \sqsubseteq B) : A, B \in \mathcal{T} \text{ with } A \not\sqsubseteq B\} \\
&\cup \{\neg(A \doteq B) : A \neq B \in \mathcal{T}\} \\
\Phi_4 :=& \{A \sqcap B = C : A, B \in \mathcal{T} \text{ and } C := A \sqcap B\} \\
\Phi_5 :=& \{\forall z.\sqsubseteq(z, A) \leftrightarrow \sqsubseteq A(z)\} \\
\Phi_6 :=& \{\forall v.\forall w.v \sqsubseteq w \to (\forall z.z \sqsubseteq v \to z \sqsubseteq w)\} \\
&\cup \{\forall v.\forall w.\forall z.z \sqsubseteq v \wedge z \sqsubseteq w \to z \sqsubseteq v \sqcap w\}
\end{aligned}$$

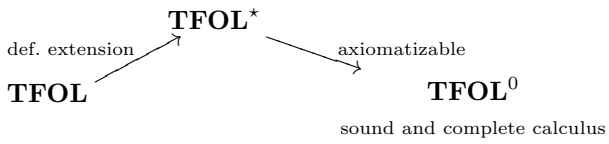
where v, w, $x : \mathbb{T}$, $z : \top$ *and* $z_A : A$ *are variable symbols* ($A \in \mathcal{T}$).

More precisely, fix a $\mathbf{TFOL}^\star$ *type hierarchy and signature. Then a* $\mathbf{TFOL}^0$ *structure satisfies* Φ *if and only if it is a* $\mathbf{TFOL}^\star$ *structure.*

Proof. First note that by definition every $\mathbf{TFOL}^\star$ type hierarchy and signature is a $\mathbf{TFOL}^0$ type hierarchy and signature, respectively. Hence, the statement of the theorem makes sense.

It is now easy (but quite lengthy) to check that each set of axioms Φ_n corresponds to the respective item n in Def. 10 ($n = 2, \ldots, 6$); items 1 and 7 already hold in Giese's logic. □

The following diagram summarizes what we have achieved so far:



The main result of this section now follows as an easy corollary.

Corollary 28 (Characterization of the existence of a sound and complete calculus for TFOL). *There exists a sound and complete calculus for* **TFOL** *if and only if the type hierarchy is Noetherian.*

Proof. In view of Thm. 23 and Lemma 21 it is sufficient to prove that there is a sound and complete calculus for $\mathbf{TFOL}^\star$ if and only if there is such a calculus for $\mathbf{TFOL}^0$. But this follows from the preceding theorem using a completely analogous construction to the one in the proof of Lemma 21. □

Remark 29. We would like to emphasize two aspects about this theorem. First, the equivalence shows that our results are *optimal*. Second, although they are phrased as existence statements, the proofs we have given are actually *constructive* in the sense that they show how to define a sound and complete calculus for **TFOL**.

This calculus is not very efficient, though (it only instantiates the **TFOL** axioms via the assumption rule from the proof of Thm. 23). In the next section we will present a more efficient tableau calculus which is still sound and complete.

3 A Tableau Calculus

Notation. We use the usual notation in our presentation of the tableau calculus below. Note that many rules are specified using *rule schemata* where *schema variables* s, t, u, ... stand for arbitrary ground terms of proper type.

In order to simplify the presentation we identify the formulae $t \doteq u$ and $u \doteq t$ and the terms $t \sqcap u$ and $u \sqcap t$, respectively.

Theorem 30 (Soundness and completeness). *The tableau calculus given in Fig. 1 and 2 is sound and complete for Noetherian type systems.*

Proof. We will only show completeness; soundness is easily verified by inspection of the individual rules.

(i) The proof of Thm. 23 essentially shows that the rules in Fig. 1 constitute a complete calculus for $\mathbf{TFOL}^0$. Indeed, we were able to modify the completeness proof from Giese [6] such that for every open tableau branch H which was saturated in a certain sense, we could construct a $\mathbf{TFOL}^0$ structure $\mathcal{S} = (\mathcal{D}, \delta, \mathcal{I})$ which was a model for H. Now recall that the domain was defined in the style of Herbrand such that it consists of normalized ground terms

$$\mathcal{D} := \mathrm{Norm}_H / \sim$$

modulo an equivalence relation

$$t \sim u :\Leftrightarrow t \doteq u \text{ or } t \doteq u \in H$$

identifying terms which are required to be equal by the tableau branch. And the interpretation of predicate symbols was defined as follows:

$$([t_1], \ldots, [t_n]) \in \mathcal{I}(p) :\Leftrightarrow p(t_1, \ldots, t_n) \in H$$

$$\frac{\varphi \wedge \psi}{\varphi,\ \psi}\ \alpha \qquad \frac{\varphi \vee \psi}{\varphi \quad \psi}\ \beta$$

$$\frac{\forall x.\varphi}{[x/t](\varphi)}\ \gamma \qquad \frac{\exists x.\varphi}{[x/c](\varphi)}\ \delta$$

with $t \in T_A$ ground, if $x : A$. with a new constant $c : A$, if $x : A$.

$$\frac{[z/t_1](\varphi),\ t_1 \doteq t_2}{[z/t_2](\varphi)}\ \text{apply-}\doteq \qquad \frac{[z/t_1](\varphi),\ t_1 \doteq t_2}{[z/(A)t_2](\varphi)}\ \text{apply-}\doteq'$$

if $\sigma(t_2) \sqsubseteq \sigma(t_1)$. where $A := \sigma(t_1)$.

$$\frac{t_1 \doteq t_2}{t_2 \sqsubseteq \sigma(t_1),\ t_1 \sqsubseteq \sigma(t_2)}\ \text{type-}\doteq \qquad \frac{}{t \sqsubseteq \sigma(t)}\ \text{type-static}$$

$$\frac{t \sqsubseteq t_1,\ t \sqsubseteq t_2}{t \sqsubseteq t_1 \sqcap t_2}\ \text{type-}\sqsubseteq\text{-}\sqcap$$

$$\frac{[z/t](\varphi),\ t \sqsubseteq A}{[z/(A)t](\varphi)}\ \text{cast-add} \qquad \frac{[z/(A)t](\varphi)}{[z/t](\varphi)}\ \text{cast-del}$$

if $A \sqsubseteq \sigma(t)$. if $\sigma(t) \sqsubseteq A$.

$$\frac{(\neg)(A)t \sqsubseteq B,\ t \sqsubseteq A}{(\neg)t \sqsubseteq B}\ \text{cast-type}$$

$$\frac{t \sqsubseteq A,\ \neg(t \sqsubseteq B)}{\square}\ \text{close-}\sqsubseteq \qquad \frac{\neg(t \doteq t)}{\square}\ \text{close-}\doteq$$

if $A \sqsubseteq B$.

$$\frac{t \sqsubseteq \bot}{\square}\ \text{close-}\bot \qquad \frac{\varphi,\ \neg\varphi}{\square}\ \text{close}$$

$$\frac{}{\varphi}\ \text{assumption}$$

if $\varphi \in \mathcal{A}$.

Fig. 1. Rules of our tableau calculus (part i)

$$\frac{}{t \sqsubseteq t}\ \text{type-refl} \qquad \frac{t \sqsubseteq u,\ u \sqsubseteq v}{t \sqsubseteq v}\ \text{type-trans}$$

$$\frac{t \sqsubseteq u,\ u \sqsubseteq t}{t \doteq u}\ \text{type-antisym}$$

$$\frac{}{\bot \sqsubseteq t}\ \text{type-}\bot \qquad \frac{}{t \sqsubseteq \top}\ \text{type-}\top$$

$$\frac{}{t \sqcap u \sqsubseteq t}\ \text{type-}\sqcap \qquad \frac{t \sqsubseteq u,\ t \sqsubseteq v}{t \sqsubseteq u \sqcap v}\ \text{type-}\sqsubseteq\text{-}\sqcap$$

$$\frac{\mathbb{T} \sqsubseteq t}{t \doteq \mathbb{T} \quad t \doteq \top}\ \text{type-}\mathbb{T}\sqsubseteq \qquad \frac{t \sqsubseteq \mathbb{T}}{t \doteq \bot \quad t \doteq \mathbb{T}}\ \text{type-}\sqsubseteq\mathbb{T}$$

$$\frac{A \doteq B}{\square}\ \text{close-type-}\doteq \qquad \frac{A \sqsubseteq B}{\square}\ \text{close-type-}\sqsubseteq$$

if $A \neq B \in \mathcal{T}$. if $A \not\sqsubseteq B$ for $A, B \in \mathcal{T}$.

$$\frac{}{A \sqsubseteq B}\ \text{type-}\sqsubseteq \qquad \frac{}{A \sqcap B \doteq C}\ \text{type-}\sqcap\text{-}\mathcal{T}$$

if $A \sqsubseteq B$ for $A, B \in \mathcal{T}$. where $C := A \sqcap B$ for $A, B \in \mathcal{T}$.

$$\frac{t \sqsubseteq x,\ x \sqsubseteq y}{t \sqsubseteq y}\ \text{type-}\sqsubseteq\text{-}\sqsubseteq$$

Fig. 2. Rules of our tableau calculus (part ii)

(see [6] for details).

(ii) Let us now suppose that the tableau branch is also saturated with respect to the rules in Fig. 2. We will now show that this implies that $\mathcal{S}$ is even a **TFOL** structure. This in turn proves completeness of our calculus.

(iii) First of all, note that Giese's construction already ensures proper interpretation of the equality and cast symbols since these have their predefined meaning already in $\mathbf{TFOL}^0$. That is, requirements 1 and 7 hold.

(iv) From saturation with respect to type-refl, type-trans, type-antisym, type-$\bot$, type-$\top$, type-$\sqcap$, type-$\sqsubseteq$-$\sqcap$, type-$\mathbb{T} \sqsubseteq$ and type-$\sqsubseteq \mathbb{T}$, it follows that $(\mathcal{D}_{\mathbb{T}}, \mathcal{I}(\sqsubseteq))$ is a type system with empty type $\mathcal{I}(\bot)$, universal type $\mathcal{I}(\top)$, intersection types given by $\mathcal{I}(\sqcap)$ and type of all types $\mathcal{I}(\mathbb{T})$. That is, requirement 2 holds as well.

(v) Requirements 3 and 4 are satisfied by saturation with respect to close-type-$\doteq$, close-type-$\sqsubseteq$, type-$\sqsubseteq$ and type-$\sqcap$-$\mathcal{T}$.

(vi) In Giese's construction the dynamic type of a domain value $[t]$ is given by its most specifically known static type $\kappa_H(t)$. We have seen in the proof of Thm. 23 that the formula $t \sqsubseteq \kappa_H(t)$ is actually on the tableau branch H. Thus, requirement 5 follows from saturation with respect to type-$\sqsubseteq$ and type-$\in$-$\sqsubseteq$.

(vii) Finally, we see that requirement 6 holds by saturation with respect to type-$\in$-$\sqsubseteq$ and type-$\in$-$\sqcap$. □

4 Applications and Future Work

Program Specification and Verification. In the context of specification and verification of programs written in a programming language with subtypes one would like to be able to reason about the type of expressions, e.g. in order to enforce dynamic type safety. As an example, suppose we are able to establish the following formula as an invariant for a system:

$$\forall t. o_1 \sqsubseteq t \leftrightarrow o_2 \sqsubseteq t$$

Then the dynamic type of two objects o_1 and o_2 is the same. In particular, if we are now able to show that some code is type-safe for o_1 then it follows that it is also type-safe for o_2. Some specification languages, such as the Java Modelling Language [10], have built-in means to express such and similar type relationships.

Generics and Noetherianity. We have seen in Rem. 4 that Java generics lead to *non-Noetherian* type hierarchies if wildcard parametrization is allowed. Nonetheless, parametric polymorphism is a major motivation behind our efforts on **TFOL**. Fortunately, it turns out that wildcard parametrization and existential types are very closely related: The proper subtypes of a wildcard type `G<?>` are just the subtypes of `G<T>` for arbitrary type `T` (cf. [11]). This suggests the following way of modeling Java type hierarchies with generics (for simplicity we assume that there is only a single generic class `G` with a single generic parameters):

1. Add all types to the type hierarchy, *except for wildcard and array types*. In particular, for every type `T` there are types `G<T>` in the type hierarchy and constant symbol $\texttt{G<T>} : \mathbb{T}$ in the signature.
2. Introduce a function symbol $\texttt{G} : \mathbb{T} \to \mathbb{T}$ and a constant symbol $\texttt{G<?>} : \mathbb{T}$.
3. Extend the axiomatization so that (i) `G<T>` and `G(T)` are identified, (ii) `G<?>` has the proper set of supertypes, and (iii) the following formula holds:

$$\forall x.\, (x \sqsubseteq \texttt{G<?>} \wedge \neg x \doteq \texttt{G<?>} \leftrightarrow \exists t.x \sqsubseteq \texttt{G}(t))$$

We thus end up with a Noetherian approximation of the original Java type hierarchy, and the results of this paper are applicable.

Dependent Types. First-class types suggest the use of functions with a type parameter instead of function families (as in the case of cast, cf. Rem. 6). But often return types or other parameters' types depend on the value of this type parameter, thus requiring the machinery of *dependent types*. For instance, a binary cast symbol should have the following signature:

$$\text{cast} : \Pi t : \mathbb{T}.\ \Pi x : \top.\ t$$

(using a syntax inspired by Martin-Löf's intuitionistic type theory [12]). In [13] Rabe has proposed a calculus for first-order logic with dependent types (**DFOL**). It should be investigated whether the benefits of **TFOL** and **DFOL** could be combined.

Completeness Gap. The calculus in Sect. 3 is also sound for the logic $\mathbf{TFOL}^{\text{fix}}$ where the domain of types is fixed to be equal to the type hierarchy. There are however formulae which hold in every such struture but cease to be tautologies with respect to **TFOL**. The structure of this resulting *completeness gap* should be investigated further since it provides information about tautologies of the fixed logic that cannot be proved using our calculus. This would allow the design of a calculus for $\mathbf{TFOL}^{\text{fix}}$ which is "sufficiently complete" for most applications.

5 Conclusion

In this paper we have introduced a statically typed logic **TFOL** in which the type hierarchy is part of the domain, and hence, types can be subjects of formulae.

Fixing the domain of types inevitably leads to an incompact logic. Therefore, we have opted for an open interpretation in which we only require the type hierarchy to be contained in the domain of types. We have shown that the set of type hierarchies for which the resulting logic has a sound and complete calculus is precisely the set of Noetherian type hierarchies. In fact, we could show a similar characterization for the logic of [6]; the result for **TFOL** then followed from an axiomatization in that logic. In Table 1 we compare the logics defined in this paper with previous work.

We have also presented a sound and complete tableau calculus which is more efficient than the one resulting directly from the axiomatization.

Table 1. Comparison of our logic with previous work

Logic	Equality	Subtyping	$\in$ and casts	First-class types	Complete calculus
Sorted **FOL** [1]		×			×
TFOL^0 [6]	×	×	×		×[1]
TFOL$^{\text{fix}}$	×	×	×	×	[2]
TFOL	×	×	×	×	×[1]

[1] for Noetherian type systems (this is optimal; Cor. 25, 28) [2] inevitably so (Thm. 18)

References

1. Schmitt, P.H., Wernecke, W.: Tableau calculus for order sorted logic. [4] 49–60
2. Weidenbach, C.: First-order tableaux with sorts. Journal of the Interest Group in Pure and Applied Logics, IGPL **3**(6) (1995) 887–906
3. Kifer, M., Wu, J.: A first-order theory of types and polymorphism in logic programming. Technical report, Department of Computer Science, University at Stony Brook (1991)
4. Bläsius, K.H., Hedtstück, U., Rollinger, C.R., eds.: Sorts and Types in Artificial Intelligence. Volume 418 of Lecture Notes in Ariticial Intelligence. Springer (1990)
5. Walther, C.: Many-sorted inferences in automated theorem proving. [4] 18–48
6. Giese, M.: A Calculus for Type Predicates and Type Coercion. In Becker, B., ed.: Tableau 2005, Springer (2005) 123–137
7. Beckert, B., Hähnle, R., Schmitt, P.H., eds.: Verification of Object-Oriented Software: The KeY Approach. LNCS 4334. Springer-Verlag (2007)
8. Giese, M.: First-order logic. [7] 21–68
9. Ulbrich, M.: Software Verification for Java 5. Diploma thesis, Universität Karlsruhe (2007)
10. Leavens, G.T., k, Poll, E., Ruby, C., Jacobs, B.: JML: Notations and tools supporting detailed design in Java. Technical Report 00-15, Department of Computer Science, Iowa State University (August 2000)
11. Cameron, N., Ernst, E., Drossopoulou, S.: Towards an Existential Types Model for Java Wildcards. In: Formal Techniques for Java-like Programs. (2007)
12. Martin-Löf, P.: Intuitionistic Type Theory. Bibliopolis (1984)
13. Rabe, F.: First-order logic with dependent types. In: Automated Reasoning, Springer (2006) 377–391