Peter H. Schmitt

First-Order Logic

Draft Version

Springer

Contents

List of Figures

Chapter 1
First-Order Logic

Peter H. Schmitt

chap:fol

1.1 Introduction

The ultimate goal of first-order logic in the context of this book, and this applies to a great extent also to Computer Science in general, is the formalization of and reasoning with natural language specifications of systems and programs. This chapter provides the logical foundations for doing so in three steps. In Section 1.2 basic first-order logic (FOL) is introduced much in the tradition of Mathematical Logic as it evolved during the 20th century as a universal theory not tailored towards a particular application area. Already this section goes beyond what is usually found in textbooks on logic for computer science in that type hierarchies are included from the start. In the short Section 1.3 two features will be added to the basic logic, that did not interest the mathematical logicians very much but are indispensable for practical reasoning. In Section 1.4 the extended basic logic will be instantiated to Java first-order logic (JFOL), tailored for the particular task of reasoning about Java programs. The focus in the present chapter is on statements; programs themselves and formulas talking about more than one program state at once will enter the scene in Chapter 2 in the KeY book.

1.2 Basic First-Order Logic

sec02:BasicFOL

1.2.1 Syntax

subsec02:BasicFOLSyntax
def:typehier

Definition 1.1. A *type hierarchy* is a pair $\mathscr{T} = (\mathrm{TSym}, \sqsubseteq)$, where

1. TSym is a set of type symbols;
2. $\sqsubseteq$ is a reflexive, transitive relation on TSym, called the subtype relation;
3. there are two designated type symbols, the *empty* type $\perp \in \mathrm{TSym}$ and the *universal* type $\top \in \mathrm{TSym}$ with $\perp \sqsubseteq A \sqsubseteq \top$ for all $A \in \mathrm{TSym}$.

We point out that no restrictions are placed on type hierarchies in contrast to other approaches requiring the existence of unique lower bounds.

Two types A, B in $\mathscr{T}$ are called *incomparable* if neither $A \sqsubseteq B$ nor $B \sqsubseteq A$.

def:Signature **Definition 1.2.** A *signature*, which is sometimes also called *vocabulary*, $\Sigma = (\mathrm{FSym}, \mathrm{PSym}, \mathrm{VSym})$ for a given type hierarchy $\mathscr{T}$ is made up of

1. a set FSym of typed function symbols, by $f : A_1 \times \ldots \times A_n \to A$ we declare the argument types of $f \in \mathrm{FSym}$ to be $A_1, \ldots, A_n$ in the given order and its result type to be A,
2. 02:item:SignaturePSym a set PSym of typed predicate symbols, by $p(A_1, \ldots, A_n)$ we declare the argument types of $p \in \mathrm{PSym}$ to be $A_1, \ldots, A_n$ in the given order, PSym obligatory contains the binary dedicated symbol $\doteq(\top, \top)$ for equality. and the two 0-place predicate symbols *true* and *false*.
3. a set VSym of typed variable symbols, by $v : A$ for $v \in \mathrm{VSym}$ we declare v to be a variable of type A.

All types A, A_i in this definition must be different from $\bot$. A 0-ary function symbol $c : \to A$ is called a constant symbol of type A. A 0-ary predicate symbol $p()$ is called a propositional variable or propositional atom. We do not allow overloading: The same symbol may not occur in $\mathrm{FSym} \cup \mathrm{PSym} \cup \mathrm{VSym}$ with different typing.

The next two definitions define by mutual induction the syntactic categories of terms and formulas of typed first-order logic.

def:TermA **Definition 1.3.** Let $\mathscr{T}$ be a type hierarchy, and Σ a signature for $\mathscr{T}$. The set Trm_A of *terms of type* A, for $A \neq \bot$, is inductively defined by

1. $v \in \mathrm{Trm}_A$ for each variable symbol $v : A \in \mathrm{VSym}$ of type A.
2. item:termComposition $f(t_1, \ldots, t_n) \in \mathrm{Trm}_A$ for each $f : A_1 \times \ldots \times A_n \to A \in \mathrm{FSym}$ and all terms $t_i \in \mathrm{Trm}_{B_i}$ with $B_i \sqsubseteq A_i$ for $1 \leq i \leq n$.
3. item:condTerm $(\text{if } \phi \text{ then } t_1 \text{ else } t_2) \in \mathrm{Trm}_A$ for $\phi \in \mathrm{Fml}$ and $t_i \in \mathrm{Trm}_{A_i}$ such that $A_2 \sqsubseteq A_1 = A$ or $A_1 \sqsubseteq A_2 = A$.

If $t \in \mathrm{Trm}_A$ we say that t is of (static) type A and write $\alpha(t) = A$.

Note, that item (2) in Definition 3 entails $c \in \mathrm{Trm}_A$ for each constant symbol $c : \to A \in \mathrm{FSym}$. Since we do not allow overloading there is for every term only one type A with $t \in \mathrm{Trm}_A$. This justifies the use of the function symbol α.

Terms of the form defined in item (3) are called *conditional terms*. They are a mere convenience. For every formula with conditional terms there is an equivalent 02MereConvenience formula without them. More liberal typing rules are possible. The theoretically most satisfying solution would be to declare the type of $(\text{if } \phi \text{ then } t_1 \text{ else } t_2)$ to be the least common supertype $A_1 \sqcup A_2$ of A_1 and A_2. But, the assumption that $A_1 \sqcup A_2$ always exists would lead to strange consequences in the program verification setting.

def:FolFml **Definition 1.4.** The set Fml of *formulas* of first-order logic for a given type hierarchy $\mathscr{T}$ and signature Σ is inductively defined as:

1. $p(t_1,\dots,t_n) \in \mathrm{Fml}$ for $p(A_1,\dots,A_n) \in \mathrm{PSym}$, and $t_i \in \mathrm{Trm}_{B_i}$ with $B_i \sqsubseteq A_i$ for all $1 \le i \le n$.
   As a consequence of item 2 in Definition 1.2 we know
   $t_1 \doteq t_2 \in \mathrm{Fml}$ for arbitrary terms t_i and *true* and *false* are in Fml.
2. $(\neg\phi)$, $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \to \psi)$, $(\phi \leftrightarrow \psi)$ are in Fml for arbitrary $\phi, \psi \in \mathrm{Fml}$.
3. $\forall v; \phi$, $\exists v; \phi$ are in Fml for $\phi \in \mathrm{Fml}$ and $v : A \in \mathrm{VSym}$.

As an inline footnote we remark that the notation for conditional terms can also be used for formulas. The *conditional formula* (if ϕ_1 then ϕ_2 else ϕ_3) is equivalent to $(\phi_1 \wedge \phi_2) \vee (\neg\phi_1 \wedge \phi_3)$.

If need arises we will make dependence of these definitions on Σ and $\mathscr{T}$ explicit by writing $\mathrm{Trm}_{A,\Sigma}$, Fml_{Σ} or $\mathrm{Trm}_{A,\mathscr{T},\Sigma}$, $\mathrm{Fml}_{\mathscr{T},\Sigma}$. When convenient we will also use the redundant notation $\forall\, A\ v; \phi$, $\exists\, A\ v; \phi$ for a variable $v : A \in \mathrm{VSym}$.

Formulas built by clause (1) only are called *atomic formulas*.

def:FreeBoundVars

Definition 1.5. For terms t and formulas ϕ we define the sets $var(t)$, $var(\phi)$ of all variables occurring in t or ϕ and the sets $fv(t), fv(\phi)$ of all variables with at least one free occurrence in t or ϕ:

$var(v) =$	$\{v\}$	$fv(v) =$	$\{v\}$	for $v \in \mathrm{VSym}$
$var(t) =$	$\bigcup_{i=1}^{n} var(t_i)$	$fv(t) =$	$\bigcup_{1=i}^{n} fv(t_i)$	for $t = f(t_1,\dots,t_n)$
$var(t) =$	$var(\phi) \cup var(t_1) \cup var(t_2)$	$fv(t) =$	$fv(\phi) \cup fv(t_1) \cup fv(t_2)$	for $t =$ (if ϕ then t_1 else t_2)
$var(\phi) =$	$\bigcup_{i=1}^{n} var(t_i)$	$fv(\phi) =$	$\bigcup_{i=1}^{n} fv(t_i)$	for $\phi = p(t_1,\dots,t_n)$
$var(\neg\phi) =$	$var(\phi)$	$fv(\neg\phi) =$	$fv(\phi)$	
$var(\phi) =$	$var(\phi_1) \cup var(\phi_2)$	$fv(\phi) =$	$fv(\phi_1) \cup fv(\phi_2)$	for $\phi = \phi_1 \circ \phi_2$ where $\circ$ is any binary Boolean operation
$var(Q\ v.\phi) = var(\phi)$		$fv(Q\ v.\phi) = var(\phi) \setminus \{v\}$		where $Q \in \{\forall, \exists\}$

A term without free variables is called a *ground term*, a formula without free variables a *ground formula* or *closed formula*.

It is an obvious consequence of this definition that every occurrence of a variable v in a term or formula with empty set of free variables is within the scope of a quantifier $Q\ v$.

One of the most important syntactical manipulations of terms and formulas are substitutions, that replace variables by terms. They will play a crucial role in proofs of quantified formulas as well as equations.

def:Substitution

Definition 1.6. A *substitution* τ is a function that associates with every variable v a type compatible term $\tau(v)$, i.e., if v is of type A then $\tau(v)$ is a term of type A' such that $A' \sqsubseteq A$.

We write $\tau = [u_1/t_1,\dots,u_n/t_n]$ to denote the substitution defined by $\mathrm{dom}(\tau) = \{u_1,\dots,u_n\}$ and $\tau(u_i) = t_i$.

A substitution τ is called a *ground substitution* if $\tau(v)$ is a ground term for all $v \in \mathrm{dom}(\tau)$.

We will only encounter substitutions τ such that $\tau(v) = v$ for all but finitely many variables v. The set $\{v \in \mathrm{VSym} \mid \tau(v) \neq v\}$ is called the *domain* of τ. It remains to make precise how a substitution τ is applied to terms and formulas.

def:applySubst

Definition 1.7. Let τ be a substitution and t a term, then $\tau(t)$ is recursively defined by:

1. $\tau(x) = x$ if $x \notin \mathrm{dom}(\tau)$
2. $\tau(x)$ as in the definition of τ if $x \in \mathrm{dom}(\tau)$
3. $\tau(f(t_1,\dots,t_k)) = f(\tau(t_1),\dots,\tau(t_k))$ if $t = f(t_1,\dots,t_k)$

Let τ be a ground substitution and ϕ a formula, then $\tau(\phi)$ is recursive defined

4. $\tau(\mathit{true}) = \mathit{true}$, $\tau(\mathit{false}) = \mathit{false}$
5. $\tau(p(t_1,\dots,t_k)) = p(\tau(t_1),\dots,\tau(t_k))$ if ϕ is the atomic formula $p(t_1,\dots,t_k)$
6. $\tau(t_1 \doteq t_k) = \tau(t_1) \doteq \tau(t_k)$
7. $\tau(\neg\phi) = \neg\tau(\phi)$
8. $\tau(\phi_1 \circ \phi_2) = \tau(\phi_1) \circ \tau(\phi_2)$ for propositional operators $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$
9. item:def:applySubstQ $\tau(Qv.\phi) = Qv.\tau_v(\phi)$ for $Q \in \{\exists, \forall\}$ and $\mathrm{dom}(\tau_v) = \mathrm{dom}(\tau) \setminus \{v\}$ with $\tau_v(x) = \tau(x)$ for $x \in \mathrm{dom}(\tau_v)$.

There are some easy conclusions from these definitions:

- If $t \in \mathrm{Trm}_A$ then $\tau(t)$ is a term of type A' with $A' \sqsubseteq A$. Indeed, if t is not a variable then $\tau(t)$ is again of type A.
- $\tau(\phi)$ meets the typing restrictions set forth in Definition 1.4.

Item 9 deserves special attention. Substitutions only act on free variables. So, when computing $\tau(Qv.\phi)$, the variable v in the body ϕ of the quantified formula is left untouched. This is effected by removing v from the domain of τ.

It is possible, and quite common, to define also the application of nonground substitutions to formulas. Care has to be taken in that case to avoid *clashes*, see Example 1.8 below. We will only need ground substitutions later on, so we sidestep this difficulty.

ex:Subst

Example 1.8. For the sake of this example we assume that there is a type symbol $\mathit{int} \in \mathrm{TSym}$, function symbols $+ : \mathit{int} \times \mathit{int} \rightarrow \mathit{int}$, $* : \mathit{int} \times \mathit{int} \rightarrow \mathit{int}$, $- : \mathit{int} \rightarrow \mathit{int}$, $\mathit{exp} : \mathit{int} \times \mathit{int} \rightarrow \mathit{int}$ and constants $0 : \mathit{int}$, $1 : \mathit{int}$, $2 : \mathit{int}$, in FSym. Definition 1.3 establishes an abstract syntax for terms. In examples we are free to use a concrete, or pretty-printing syntax. Here we use the familiar notation $a+b$ instead of $+(a,b)$, $a*b$ or ab instead of $*(a,b)$, and a^b instead of $\mathit{exp}(a,b)$. Let furthermore $x : \mathit{int}$, $y : \mathit{int}$ be variables of sort int. The following table shows the results of applying the substitution $\tau_1 = [x/0, y/1]$ to the given formulas

$$\begin{array}{ll}
\phi_1 = \forall x; ((x+y)^2 \doteq x^2 + 2xy + y^2) & \tau_1(\phi_1) = \forall x; ((x+1)^2 \doteq x^2 + 2*x*1 + 1^2) \\
\phi_2 = (x+y)^2 \doteq x^2 + 2xy + y^2 & \tau_1(\phi_2) = (0+1)^2 \doteq 0^2 + 2*0*1 + 1^2 \\
\phi_3 = \exists x; (x > y) & \tau_1(\phi_3) = \exists x; (x > 1)
\end{array}$$

Application of the nonground substitution $\tau_2 = [y/x]$ on ϕ_3 leads to $\exists x; (x > x)$. While $\exists x; (x > y)$ is true for all assignments to y the substituted formula $\tau(\phi_3)$ is not. Validity is preserved if we restrict to clash-free substitutions. A substitution τ is said to create a *clash* with formula ϕ if a variable w in a term $\tau(v)$ for $v \in \mathrm{dom}(\tau)$ ends up in the scope of a quantifier Qw in ϕ. For τ_2 the variable x in $\tau_2(y)$ will end up in the scope of $\forall x$;

The concept of a substitution also comes in handy to solve the following notational problem. Let ϕ be a formula that contains somewhere an occurrence of the term t_1. How should we refer to the formula arising from ϕ by replacing t_1 by t_2? E.g. replace $2xy$ in ϕ_2 by $xy2$. The solution is to use a new variable z and a formula ϕ_0 such that $\phi = [z/t_1]\phi_0$. Then the replaced formula can be referred to as $[z/t_2]\phi_0$. In the example we would have $\phi_0 = (x+y)^2 \doteq x^2+z+y^2$. This trick will be extensively used in Figure 1.1 and 1.2.

We come back to the conditinal terms introduced in item (3) of Definition 1.3. As mentioned on page 2 they do not increase the expressive power of the language. This claim is now proved in the following lemma.

elimCondTerm

Lemma 1.9. *For any formula ϕ there is a logically equivalent formula ϕ_1 that does not contain conditional terms.*

Proof. Let ϕ be a formula containing an occurrence of the conditional term $t =$ (if ψ then t_1 else t_2). We may assume without loss of generality that ϕ is in prenex normal form, i.e. $\phi = Q_1\ v_1.\ldots Q_n\ v_n.\phi_0$ with ϕ_0 a quantifierfree formula.

Let ϕ_0^1 be the formula arising from ϕ_0 by replacing this occurrence of t by t_1 and ϕ_0^2 the formula arising from ϕ_0 by replacing the occurrence of t by t_2. Let

$$\phi_0' = (\psi \land \phi_0^1) \lor (\psi \land \phi_0^2)$$

It can be easily seen that ϕ is logically equivalent to $Q_1\ v_1.\ldots Q_n\ v_n.\phi_0'$ and $Q_1\ v_1.\ldots Q_n\ v_n.\phi_0'$ has one occurrence of a conditional term less than ϕ. Iterating this construction we arrive at an equivalent formula without conditional terms. □

1.2.2 Calculus

subsec02:BasicFOLCalculus

The main reason nowadays for introducing a formal, machine readable syntax for formulas, as we did in the previous subsection, is to get machine support for logical reasoning. For this, one needs first a suitable calculus and then an efficient implementation. In this subsection we present the rules for basic first-order logic. A machine readable representation of these rules will be covered in Chapter 4 in the KeY book. Chapter 15 in the KeY book provides an unhurried introduction on using the KeY theorem prover based on these rules that can be read without prerequisites. So the reader may want to step through it before continuing here.

The calculus of our choice is the *sequent calculus*. The basic data that is manipulated by the rules of the sequent calculus are *sequents*. These are of the form $\phi_1,\ldots,\phi_n \Longrightarrow \psi_1,\ldots,\psi_m$. The formulas $\phi_1,\ldots,\phi_n$ at the left-hand side of the sequent separator $\Longrightarrow$ are the antecedents of the sequent; the formulas $\psi_1,\ldots,\psi_m$ on

$$\mathsf{andLeft}\ \frac{\Gamma,\phi,\psi \Longrightarrow \Delta}{\Gamma,\phi\wedge\psi \Longrightarrow \Delta} \qquad \mathsf{andRight}\ \frac{\Gamma \Longrightarrow \phi,\Delta \qquad \Gamma \Longrightarrow \psi,\Delta}{\Gamma \Longrightarrow \phi\wedge\psi,\Delta}$$

$$\mathsf{orRight}\ \frac{\Gamma \Longrightarrow \phi,\psi,\Delta}{\Gamma \Longrightarrow \phi\vee\psi,\Delta} \qquad \mathsf{orLeft}\ \frac{\Gamma,\phi \Longrightarrow \Delta \qquad \Gamma,\psi \Longrightarrow \Delta}{\Gamma,\phi\vee\psi \Longrightarrow \Delta}$$

$$\mathsf{impRight}\ \frac{\Gamma,\phi \Longrightarrow \psi,\Delta}{\Gamma \Longrightarrow \phi\to\psi,\Delta} \qquad \mathsf{impLeft}\ \frac{\Gamma \Longrightarrow \phi,\Delta \qquad \Gamma,\psi \Longrightarrow \Delta}{\Gamma,\phi\to\psi \Longrightarrow \Delta}$$

$$\mathsf{notLeft}\ \frac{\Gamma \Longrightarrow \phi,\Delta}{\Gamma,\neg\phi \Longrightarrow \Delta} \qquad \mathsf{notRight}\ \frac{\Gamma,\phi \Longrightarrow \Delta}{\Gamma \Longrightarrow \neg\phi,\Delta}$$

$$\mathsf{allRight}\ \frac{\Gamma \Longrightarrow [x/c](\phi),\Delta}{\Gamma \Longrightarrow \forall x;\phi,\Delta} \qquad \mathsf{allLeft}\ \frac{\Gamma,\forall x;\phi,[x/t](\phi) \Longrightarrow \Delta}{\Gamma,\forall x;\phi \Longrightarrow \Delta}$$

with $c : \to A$ a new constant, if $x{:}A$ with $t \in \mathrm{Trm}_{A'}$ ground, $A' \sqsubseteq A$, if $x{:}A$

$$\mathsf{exLeft}\ \frac{\Gamma,[x/c](\phi) \Longrightarrow \Delta}{\Gamma,\exists x;\phi \Longrightarrow \Delta} \qquad \mathsf{exRight}\ \frac{\Gamma \Longrightarrow \exists x;\phi,[x/t](\phi),\Delta}{\Gamma \Longrightarrow \exists x;\phi,\Delta}$$

with $c : \to A$ a new constant, if $x{:}A$ with $t \in \mathrm{Trm}_{A'}$ ground, $A' \sqsubseteq A$, if $x{:}A$

$$\mathsf{close}\ \frac{*}{\Gamma,\phi \Longrightarrow \phi,\Delta}$$

$$\mathsf{closeFalse}\ \frac{*}{\Gamma,\mathrm{false} \Longrightarrow \Delta} \qquad \mathsf{closeTrue}\ \frac{*}{\Gamma \Longrightarrow \mathrm{true},\Delta}$$

[illegible] **Fig. 1.1** First-order rules for the logic FOL fig:folrules

the right are the succedents. In our version of the calculus antecedent and succedent are sets of formulas, i.e., the order and multiple occurrences are not relevant. Furthermore, we will assume that all ϕ_i and ψ_j are ground formulas. A sequent $\phi_1,\ldots,\phi_n \Longrightarrow \psi_1,\ldots,\psi_m$ is valid iff the formula $\bigwedge_{1=i}^{n}\phi_i \to \bigvee_{1=j}^{m}\psi_j$ is valid.

The concept of sequent calculi was introduce by the German logician Gerhard Gentzen in the 1930s, though for a very different purpose.

Figures 1.1 and 1.2 show the usual set of rules of the sequent calculus with equality as it can be found in many text books, e.g. Gallier87 [Gallier, 1987, Section 5.4]. Rules are written in the form

$$\mathsf{ruleName}\ \frac{P_1,\ldots P_n}{C}$$

The P_i is called the *premisses* and C the *conclusion* of the rule. There is no theoretical limit on n, but most of the time $n = 1$, sometimes $n = 2$, and in rare cases $n = 3$. Note, that premiss and conclusion contain the schematic variables Γ,Δ for set of formulas, ψ,ϕ for formulas and t,c for terms and constants. We use Γ,ϕ and ψ,Δ to stand for $\Gamma\cup\{\phi\}$ and $\{\psi\}\cup\Delta$. An instance of a rule is obtained by consistently replacing the schematic variables in premiss and conclusion by the corresponding entities: sets of formulas, formulas, etc. Rule application in KeY proceeds from bottom to top. Suppose we want to prove a sequent s_2. We look for a rule R such that there is an instantiation *Inst* of the schematic variables in R such that the instantiation of its conclusion $Inst(S_2)$ equals s_2. After rule application we are left with the task to prove the sequent $Inst(S_1)$. If S_1 is empty, we succeeded.

defi:closingRules

Definition 1.10. The rules close, closeFalse, and closeTrue from Figure 1.1 are called *closing rules* since their premisses are empty.

Since there are rules with more than one premiss the proof process sketched above will result in a proof tree.

defi:proofTree

Definition 1.11. A *proof tree* is a tree, shown with the root at the bottom, such that

1. each node is labeled with a sequent or the symbol $*$,
2. if an inner node n is annotated with $\Gamma \Longrightarrow \Delta$ then there is an instance of a rule whose conclusion is $\Gamma \Longrightarrow \Delta$ and the child node, or children nodes of n are labeled with the premiss or premisses of the rule instance.

A branch in a proof tree is called *closed* if its leaf is labeled by $*$. A proof tree is called *closed* if all its branches are closed, or equivalently if all its leaves are labeled with $*$.

We say that a sequent $\Gamma \Longrightarrow \Delta$ can be derived if there is a closed proof tree whose root is labeled by $\Gamma \Longrightarrow \Delta$.

As a first simple example, we will derive the sequent $\Longrightarrow p \land q \to q \land p$. The same formula is also used in the explanation of the KeY prover in Chapter 15 in the KeY book. As its antecedent is empty, this sequent says that the propositional formula $p \land q \to q \land p$ is a tautology. Application of the rule impRight reduces our proof goal to $p \land q \Longrightarrow q \land p$ and application of andLeft further to $p,q \Longrightarrow q \land p$. Application of andRight splits the proof into the two goals $p,q \Longrightarrow q$ and $p,q \Longrightarrow p$. Both goals can be discharged by an application of the close rule. The whole proof can concisely be summarized as a tree

$$\cfrac{\cfrac{\cfrac{\cfrac{*}{p,q \Longrightarrow q} \qquad \cfrac{*}{p,q \Longrightarrow p}}{p,q \Longrightarrow q \land p}}{p \land q \Longrightarrow q \land p}}{\Longrightarrow p \land q \to q \land p}$$

Let us look at an example derivation involving quantifiers. If you are puzzled by the use of substitutions $[x/t]$ in the formulations of the rules you should refer back to Example 1.8. We assume that $p(A,A)$ is a binary predicate symbol with both arguments of type A. Here is the, nonbranching, proof tree for the formula $\exists v; \forall w; p(v,w) \to \forall w; \exists v; p(v,w)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{*}{\forall w; p(c,w), p(c,d) \Longrightarrow p(c,d), \exists v; p(v,d)}}{\forall w; p(c,w) \Longrightarrow \exists v; p(v,d)}}{\exists v; \forall w; p(v,w) \Longrightarrow \forall w; \exists v; p(v,w)}}{\Longrightarrow \exists v; \forall w; p(v,w) \to \forall w; \exists v; p(v,w)}$$

The derivation starts, from bottom to top, with the rule impRight. The next line above is obtained by applying exLeft and allRight. This introduces new constant symbols $c :\to A$ and $d :\to A$. The top line is obtained by the rules exRight and allLeft with the ground substitutions $[w/d]$ and $[v/c]$. The proof terminates by an application of close resulting in an empty proof obligation. An application of the rules exLeft, allRight is often called *Skolemization* and the new constant symbols called *Skolem constants*.

$$\text{eqLeft}\ \dfrac{\Gamma, t_1 \doteq t_2, [z/t_1](\phi), [z/t_2](\phi) \Longrightarrow \Delta}{\Gamma, t_1 \doteq t_2, [z/t_1](\phi) \Longrightarrow \Delta}$$
provided $\sigma(t_1) \sqsubseteq \sigma(t_2)$

$$\text{eqRight}\ \dfrac{\Gamma, t_1 \doteq t_2 \Longrightarrow [z/t_2](\phi), [z/t_1](\phi), \Delta}{\Gamma, t_1 \doteq t_2 \Longrightarrow [z/t_2](\phi), \Delta}$$
provided $\sigma(t_2) \sqsubseteq \sigma(t_1)$

$$\text{eqSymmLeft}\ \dfrac{\Gamma, t_2 \doteq t_1 \Longrightarrow \Delta}{\Gamma, t_1 \doteq t_2 \Longrightarrow \Delta} \qquad \text{eqReflLeft}\ \dfrac{\Gamma, t \doteq t \Longrightarrow \Delta}{\Gamma \Longrightarrow \Delta}$$

$$\text{eqDynamicSort}\ \dfrac{\Gamma, t_1 \doteq t_2, \exists x(x \doteq t_1 \land x \doteq t_2) \Longrightarrow \Delta}{\Gamma, t_1 \doteq t_2 \Longrightarrow \Delta}$$
if $\sigma(t_1)$ and $\sigma(t_2)$ are incomparable,
the sort C of x is new and satisfies
$C \sqsubset \sigma(t_1)$ and $C \sqsubset \sigma(t_2)$

rule:eqLeft rule:eqRight

Fig. 1.2 Equality rules for the logic FOL

fig:eqrules

The calculus implemented in the KeY system is less liberal with regard to the syntactically allowed equations. In contrast to Definition 1.2 in this technical report an equation $t_1 \doteq t_2$ is in the implemented logic only allowed if the types of t_1 and t_2 are compatible. Thus rule eqDynamicSort in Figure 1.2 would never be applicable. This rule is however needed for completeness of liberal the variant presented here. Further comments in the relationship between the logic presented here and the logic implemented in KeY will be given at the end of subsection 1.2.5.

The rules involving equality are shown in Figure 1.2. The rules eqLeft and eqRight formalize the intuitive application of equations: if $t_1 \doteq t_2$ is known, we may replace wherever we want t_1 by t_2. In typed logic the formula after substitution might not be well-typed. Here is an example for the rule eqLeft without restriction. Consider two types $A \neq B$ with $B \sqsubseteq A$, two constant symbols $a : \to A$ and $b : \to B$, and a unary predicate $p(B)$. Applying unrestricted eqLeft on the sequent $b \doteq a, p(b) \Longrightarrow$ would result in $b \doteq a, p(b), p(a) \Longrightarrow$. There is in a sense logically nothing wrong with this, but $p(a)$ is not well-typed. This motivates the provisions in the rules eqLeft and eqRight.

Let us consider a short example of equational reasoning involving the function symbol $+ : int \times int \to int$.

7 $*$

6 $(a+(b+c))+d) \doteq a+((b+c)+d), \forall x,y,z;((x+y)+z \doteq x+(y+z))$
$(b+c)+d \doteq b+(c+d), a+(b+c))+d) \doteq a+(b+(c+d)) \Longrightarrow$
$(a+(b+c))+d) \doteq a+(b+(c+d))$

5 $(a+(b+c))+d) \doteq a+((b+c)+d), \forall x,y,z;((x+y)+z \doteq x+(y+z))$
$(b+c)+d \doteq b+(c+d) \Longrightarrow$
$(a+(b+c))+d) \doteq a+(b+(c+d))$

4 $(a+(b+c))+d) \doteq a+((b+c)+d), \forall x,y,z;((x+y)+z \doteq x+(y+z)) \Longrightarrow$
$(a+(b+c))+d) \doteq a+(b+(c+d))$

3 $\forall x,y,z;((x+y)+z \doteq x+(y+z)) \Longrightarrow (a+(b+c))+d \doteq a+(b+(c+d))$

2 $\forall x,y,z;((x+y)+z \doteq x+(y+z)) \Longrightarrow$
$\forall x,y,z,u;(((x+(y+z))+u) \doteq x+(y+(z+u)))$

1 $\Longrightarrow \forall x,y,z;((x+y)+z \doteq x+(y+z)) \to$
$\forall x,y,z,u;(((x+(y+z))+u) \doteq x+(y+(z+u)))$

Line 1 states the proof goal, a consequence from the associativity of $+$. Line 2 is obtained by an application of impRight while line 3 results from a four-fold application of allRight introducing the new constant symbol a, b, c, d for the universally quantified variables x, y, z, u, respectively. Line 4 in turn is arrived at by an application of allLeft with the substitution $[x/a, y/(b+c), z/d]$. Note, that the universally quantified formula does not disappear. In Line 5 another application of allLeft, but this time with the substitution $[x/b, y/c, z/d]$, adds the equation $(b+c)+d \doteq b+(c+d)$ to the antecedent. Now, eqLeft is applicable, replacing on the left-hand side of the sequent the term $(b+c)+d$ in $(a+b)+(c+d) \doteq a+(b+(c+d))$ by the right-hand side of the equation $(b+c)+d \doteq b+(c+d)$. This results in the same equation as in the succedent. Rule close can thus be applied.

Already this small example reveals the technical complexity of equational reasoning. Whenever the terms involved in equational reasoning are of a special type one would prefer to use decision procedures for the relevant specialized theories, e.g., for integer arithmetic or the theory of arrays.

We will see in the next section, culminating in Theorem 1.21, that the rules from Figures 1.1 and 1.2 are sufficient with respect to the semantics to be introduced in that section. But, it would be very inefficient to base proofs only on these first principles. The KeY system contains many derived rules to speed up the proof process. Let us just look at one randomly chosen example:

$$\mathsf{doubleImpLeft}\ \frac{\Gamma \Longrightarrow b,\Delta \qquad \Gamma \Longrightarrow c,\Delta \qquad \Gamma, d \Longrightarrow \Delta}{\Gamma, b \to (c \to d) \Longrightarrow \Delta}$$

It is easy to see that doubleImpLeft can be derived.

There is one more additional rule that we should not fail to mention:

$$\mathsf{cut}\ \frac{\Gamma \Longrightarrow \phi,\Delta \qquad \Gamma,\phi \Longrightarrow \Delta}{\Gamma \Longrightarrow \Delta}$$
provided ϕ is a ground formula

On the basis of the notLeft rule this is equivalent to

$$\mathsf{cut'}\ \frac{\Gamma,\neg\phi \Longrightarrow \Delta \qquad \Gamma,\phi \Longrightarrow \Delta}{\Gamma \Longrightarrow \Delta}$$
provided ϕ is a ground formula

It becomes apparent that the cut rule allows at any node in the proof tree proceeding by a case distinction. This is the favorite rule for user interaction. The system might not find a proof for $\Gamma \Longrightarrow \Delta$ automatically, but for a cleverly chosen ϕ automatic proofs for both $\Gamma,\phi \Longrightarrow \Delta$ and $\Gamma \Longrightarrow \phi,\Delta$ might be possible.

1.2.3 Semantics

subsec02:BasicFOLSemantics

So far we trusted that the logical rules contained in Figures 1.1 and 1.2 are self-evident. In this section we provide further support that the rules and the deduction system as a whole are sound, in particular no contradiction can be derived. So far we also had only empirical evidence that the rules are sufficient. The semantical approach presented in this section will open up the possibility to rigorously prove completeness.

def:FolUniverse

Definition 1.12. A *universe* or *domain* for a given type hierarchy $\mathscr{T}$ and signature Σ consists of

1. a set D,
2. a typing function $\delta : D \to \mathrm{TSym} \setminus \{\bot\}$ such that for every $A \in \mathrm{TSym}$ the set $D^A = \{d \in D \mid \delta(d) \sqsubseteq A\}$ is not empty.

The set $D^A = \{d \in D \mid \delta(d) \sqsubseteq A\}$ is called the type universe or type domain for A. Definition 1.12 implies that for different types $A, B \in \mathrm{TSym} \setminus \{\bot\}$ there is an element $o \in D^A \cap D^B$ only if there exists $C \in \mathrm{TSym}$, $C \neq \bot$ with $C \sqsubseteq A$ and $C \sqsubseteq B$.

lem:TypeDomain

Lemma 1.13. *The type domains for a universe (D,δ) share the following properties*

1. $D^{\bot} = \emptyset$, $D^{\top} = D$,
2. $D^A \subseteq D^B$ if $A \sqsubseteq B$,

3. $D^C = D^A \cap D^B$ in case the greatest lower bound C of A and B exists.

def:FolStructure **Definition 1.14.** A first-order *structure* $\mathscr{M}$ for a given type hierarchy $\mathscr{T}$ and signature Σ consists of

- a domain (D,δ),
- an interpretation I

such that

item:FSymInterp 1. $I(f)$ is a function from $D^{A_1} \times \cdots \times D^{A_n}$ into D^A for $f : A_1 \times \ldots \times A_n \to A$ in FSym,
item:PSymInterp 2. $I(p)$ is a subset of $D^{A_1} \times \cdots \times D^{A_n}$ for $p(A_1,\ldots,A_n)$ in PSym,
3. $I(\doteq) = \{(d,d) \mid d \in D\}$.

For constant symbols $c : \to A \in$ FSym requirement (1) reduces to $I(c) \in D^A$. It has become customary to interpret an empty product as the set $\{\emptyset\}$, where $\emptyset$ is deemed to stand for the empty tuple. Thus requirement (2) reduces for $n = 0$ to $I(p) \subseteq \{\emptyset\}$. Only if need arises, we will say more precisely that $\mathscr{M}$ is a $\mathscr{T}$-Σ-structure.

def:VarAssignment **Definition 1.15.** Let $\mathscr{M}$ be a first-order structure with universe D.

A *variable assignment* is a function β : VSym $\to D$ such that $\beta(v) \in D^A$ for $v : A \in$ VSym.

For a variable assignment β, a variable $v : A \in$ VSym and a domain element $d \in D^A$, the following definition of a modified assignment will be needed later on:

$$\beta_v^d(v') = \begin{cases} d & \text{if } v' = v \\ \beta(v') & \text{if } v' \neq v \end{cases}$$

The next two definitions define the evaluation of terms and formulas with respect to a structure $\mathscr{M} = (D,\delta,I)$ for given type hierarchy $\mathscr{T}$, signature Σ, and variable assignment β by mutual recursion.

def:TermEval **Definition 1.16.** For every term $t \in \mathrm{Trm}_A$, we define its evaluation $\mathrm{val}_{\mathscr{M},\beta}(t)$ inductively by:

- $\mathrm{val}_{\mathscr{M},\beta}(v) = \beta(v)$ for any variable v.
- $\mathrm{val}_{\mathscr{M},\beta}(f(t_1,\ldots,t_n)) = I(f)(\mathrm{val}_{\mathscr{M},\beta}(t_1),\ldots,\mathrm{val}_{\mathscr{M},\beta}(t_n))$.
- $\mathrm{val}_{\mathscr{M},\beta}(\text{if } \phi \text{ then } t_1 \text{ else } t_2) = \begin{cases} \mathrm{val}_{\mathscr{M},\beta}(t_1) & \text{if } (\mathscr{M},\beta) \models \phi \\ \mathrm{val}_{\mathscr{M},\beta}(t_2) & \text{if } (\mathscr{M},\beta) \not\models \phi \end{cases}$

def:ForEval **Definition 1.17.** For every formula $\phi \in$ Fml, we define when ϕ is considered to be true with respect to $\mathscr{M}$ and β, which is denoted with $(\mathscr{M},\beta) \models \phi$, by:

$$\begin{array}{lll}
1\ (\mathcal{M},\beta) \models true, & (\mathcal{M},\beta) \not\models false & \\
2\ (\mathcal{M},\beta) \models p(t_1,\ldots,t_n) & \text{iff} & (\mathrm{val}_{\mathcal{M},\beta}(t_1),\ldots,\mathrm{val}_{\mathcal{M},\beta}(t_n)) \in I(p) \\
3\ (\mathcal{M},\beta) \models \neg\phi & \text{iff} & (\mathcal{M},\beta) \not\models \phi \\
4\ (\mathcal{M},\beta) \models \phi_1 \wedge \phi_2 & \text{iff} & (\mathcal{M},\beta) \models \phi_1 \text{ and } (\mathcal{M},\beta) \models \phi_2 \\
5\ (\mathcal{M},\beta) \models \phi_1 \vee \phi_2 & \text{iff} & (\mathcal{M},\beta) \models \phi_1 \text{ or } (\mathcal{M},\beta) \models \phi_2 \\
6\ (\mathcal{M},\beta) \models \phi_1 \to \phi_2 & \text{iff} & (\mathcal{M},\beta) \not\models \phi_1 \text{ or } (\mathcal{M},\beta) \models \phi_2 \\
7\ (\mathcal{M},\beta) \models \phi_1 \leftrightarrow \phi_2 & \text{iff} & ((\mathcal{M},\beta) \models \phi_1 \text{ and } (\mathcal{M},\beta) \models \phi_2) \text{ or} \\
 & \text{iff} & ((\mathcal{M},\beta) \not\models \phi_1 \text{ and } (\mathcal{M},\beta) \not\models \phi_2) \\
8\ (\mathcal{M},\beta) \models \forall\, A\ v;\phi & \text{iff} & (\mathcal{M},\beta_v^d) \models \phi \text{ for all } d \in D^A \\
9\ (\mathcal{M},\beta) \models \exists\, A\ v;\phi & \text{iff} & (\mathcal{M},\beta_v^d) \models \phi \text{ for at least one } d \in D^A
\end{array}$$

For a 0-place predicate symbol p, clause (2) says $\mathcal{M} \models p$ iff $\emptyset \in I(p)$. Thus the interpretation I acts in this case as an assignment of truth values to p. This explains why we have called 0-place predicate symbols propositional atoms.

Given the restriction on $I(\doteq)$ in Definition 1.14, clause (2) also says $(\mathcal{M},\beta) \models t_1 \doteq t_2$ iff $\mathrm{val}_{\mathcal{M},\beta}(t_1) = \mathrm{val}_{\mathcal{M},\beta}(t_2)$.

For a set Φ of formulas, we use $(\mathcal{M},\beta) \models \Phi$ to mean $(\mathcal{M},\beta) \models \phi$ for all $\phi \in \Phi$.

If ϕ is a formula without free variables, we may write $\mathcal{M} \models \phi$ since the variable assignment β is not relevant here.

To prepare the ground for the next definition we explain the concept of extensions between type hierarchies.

defi:THextention **Definition 1.18.** A type hierarchy $\mathcal{T}_2 = (\mathrm{TSym}_2, \sqsubseteq_2)$ is an *extension* of a type hierarchy $\mathcal{T}_1 = (\mathrm{TSym}_1, \sqsubseteq_1)$, in symbols $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$, if

1. $\mathrm{TSym}_1 \subseteq \mathrm{TSym}_2$
2. $\sqsubseteq_2$ is the smallest subtype relation containing $\sqsubseteq_1 \cup \Delta$ where Δ is a set of pairs (S,T) with $T \in \mathrm{TSym}_1$ and $S \in \mathrm{TSym}_2 \setminus \mathrm{TSym}_1$.

So, new types can only be declared to be subtypes of old types, never supertypes. Also, $\bot \sqsubseteq_2 A \sqsubseteq_2 \top$ for all new types A.

Definition 1.18 forbids the introduction of subtype chains like $A \sqsubseteq B \sqsubseteq T$ into the type hierarchy. However, it can be shown that relaxing the definition in that respect results in an equivalent notion of logical consequence. We keep the restriction here since it simplifies reasoning about type hierarchy extensions.

For later reference, we note the following lemma.

lem:THExt **Lemma 1.19.** *Let $\mathcal{T}_2 = (\mathrm{TSym}_2, \sqsubseteq_2)$ be an extension of $\mathcal{T}_1 = (\mathrm{TSym}_1, \sqsubseteq_1)$ with $\sqsubseteq_2$ the smallest subtype relation containing $\sqsubseteq_1 \cup \Delta$, for some $\Delta \subseteq (\mathrm{TSym}_2 \setminus \mathrm{TSym}_1) \times \mathrm{TSym}_1$.*

Then, for $A, B \in \mathrm{TSym}_1$, $C \in \mathrm{TSym}_2 \setminus \mathrm{TSym}_1$, $D \in \mathrm{TSym}_2$

item:THExt-1 *1. $A \sqsubseteq_2 B$ iff $A \sqsubseteq_1 B$*
item:THExt-2 *2. $C \sqsubseteq_2 A$ iff $T \sqsubseteq_1 A$ for some $(C,T) \in \Delta$.*
3. $D \sqsubseteq_2 C$ iff $D = C$ or $D = \bot$

Proof. This follows easily from the fact that no supertype relations of the form $A \sqsubseteq_2 C$ for new type symbols C are stipulated. □

def:LogicConsequence

Definition 1.20. Let $\mathcal{T}$ be a type hierarchy and Σ a signature, $\phi \in \mathrm{Fml}_{\mathcal{T},\Sigma}$ a formula without free variables, and $\Phi \subseteq \mathrm{Fml}_{\mathcal{T},\Sigma}$ a set of formulas without free variables.

1. ϕ is a *logical consequence* of Φ, in symbols $\Phi \models \phi$, if for all type hierarchies $\mathcal{T}'$ with $\mathcal{T} \sqsubseteq \mathcal{T}'$ and all $\mathcal{T}'$-Σ-structures $\mathcal{M}$ such that $\mathcal{M} \models \Phi$, also $\mathcal{M} \models \phi$ holds.
2. ϕ is *universally valid* if it is a logical consequence of the empty set, i.e., if $\emptyset \models \phi$.
3. ϕ is satisfiable if there is a type hierarchy $\mathcal{T}'$, with $\mathcal{T} \sqsubseteq \mathcal{T}'$ and a $\mathcal{T}'$-Σ-structure $\mathcal{M}$ with $\mathcal{M} \models \phi$.

The extension of Definition 1.20 to formulas with free variables is conceptually not difficult but technically a bit involved. The present definition covers however all we need in this book.

The central concept is universal validity since, for finite Φ, it can easily be seen that:

- $\Phi \models \phi$ iff the formula $\bigwedge \Phi \to \phi$ is universally valid.
- ϕ is satisfiable iff $\neg\phi$ is not universally valid.

The notion of *logical consequence* from Definition 1.20 is sometimes called *super logical consequence* to distinguish it from the concept $\Phi \models_{\mathcal{T},\Sigma} \phi$ denoting that for any $\mathcal{T}$-Σ-structure $\mathcal{M}$ with $\mathcal{M} \models \Phi$ also $\mathcal{M} \models \phi$ is true.

To see the difference, let the type hierarchy $\mathcal{T}_1$ contain types A and B such that the greatest lower bound of A and B is $\perp$. For the formula $\phi_1 = \forall\, A\, x; (\forall\, B\, y; (x \neq y))$ we have $\models_{\mathcal{T}_1} \phi_1$. Let $\mathcal{T}_2$ be the type hierarchy extending $\mathcal{T}_1$ by a new type D and the ordering $D \sqsubseteq A$, $D \sqsubseteq B$. Now, $\models_{\mathcal{T}_2} \phi_1$ does no longer hold true.

The phenomenon that the tautology property of a formula ϕ depends on symbols that do not occur in ϕ is highly undesirable. This is avoided by using the logical consequence defined as above. In this case we have $\not\models \phi_1$.

thm:FOLcompleteness

Theorem 1.21 (Soundness and Completeness Theorem). *Let $\mathcal{T}$ be a type hierarchy and Σ a signature, $\phi \in \mathrm{Fml}_{\mathcal{T},\Sigma}$ without free variables. The calculus for FOL is given by the rules in Figures 1.1 and 1.2. Assume that for every type $A \in \mathcal{T}$ there is a constant symbol of type A' with $A' \sqsubseteq A$.*

Then:

- *if there is a closed proof tree in FOL for the sequent $\Longrightarrow \phi$ then ϕ is universally valid*
  i.e., FOL is sound.
- *if ϕ is universally valid then there is a closed proof tree for the sequent $\Longrightarrow \phi$ in FOL.*
  i.e., FOL is complete.

For the untyped calculus a proof of the sound- and completeness theorem may be found in any decent text book, e.g. Gallier87 [Gallier, 1987, Section 5.6]. Giese05 Giese [2005] covers the typed version in a setting with additional cast functions and type predicates. His proof does not consider super logical consequence and requires that type hierarchies are lower-semi-lattices.

Concerning the constraint placed on the signature in Theorem 1.21, the calculus implemented in the KeY system takes a slightly different but equivalent approach: instead of requiring the existence of sufficient constants, it allows one to derive via the rule ex_unused, for every $A \in \mathscr{T}$ the formula $\exists x(x \doteq x)$, with x a variable of type A.

def:soundRules

Definition 1.22. A rule

$$\frac{\Gamma_1 \Longrightarrow \Delta_1 \qquad \Gamma_2 \Longrightarrow \Delta_2}{\Gamma \Longrightarrow \Delta}$$

of a sequent calculus is called

- *sound* if whenever $\Gamma_1 \Longrightarrow \Delta_1$ and $\Gamma_2 \Longrightarrow \Delta_2$ are universally valid so is $\Gamma \Longrightarrow \Delta$.
- *complete* if whenever $\Gamma \Longrightarrow \Delta$ is universally valid then also $\Gamma_1 \Longrightarrow \Delta_1$ and $\Gamma_2 \Longrightarrow \Delta_2$ are universally valid.

For nonbranching rules and rules with side conditions the obvious modifications have to be made.

An inspection of the proof of Theorem 1.21 shows that if all rules of a calculus are sound then the calculus itself is sound. This is again stated as Lemma 4.7 in Section 4.4 in the KeY book devoted to the soundness management of the KeY system. In the case of soundness also the reverse implication is true: if a calculus is sound then all its rules will be sound.

The inspection of the proof of Theorem 1.21 also shows that the calculus is complete if all its rules are complete. This criterion is however not necessary, a complete calculus may contain rules that are not complete.

We skipped the proof of Lemma 1.13 to not interupt the follow of the presentation. We deliver it here

Proof. Proof of Lemma 1.13:

1. We start with the definition $D^{\perp} = \{d \in D \mid \delta(d) \sqsubseteq \perp\}$. Since $\perp$ is the only type in TSym with $\perp \sqsubseteq \perp$ and $\perp$ is not in the range of δ we get $D^{\perp} = \emptyset$.
   For the second claim we start again with the definition $D^{\top} = \{d \in D \mid \delta(d) \sqsubseteq \top\}$. Since $A \sqsubseteq \top$ for any type A we get $D^{\top} = D$.
2. Assume $A \sqsubseteq B$. If $d \in D^A$ then $\delta(d) \sqsubseteq A$. Transitivity of the subtype relation entails $\delta(d) \sqsubseteq B$ and thus $d \in D^B$.
3. If C is the greatest lower bound of A and B we have $C \sqsubseteq A$ and $C \sqsubseteq B$ and thus by (2) $D^C \subseteq D^A$ and $D^C \subseteq D^B$. Therefore, $D^C \subseteq D^A \cap D^B$. If $d \in D^A \cap D^B$ then $\delta(d) \sqsubseteq A$ and $\delta(d) \sqsubseteq B$. Thus $\delta(d) \sqsubseteq C$ and therefore $d \in D^C$.

□

1.2.4 Digression on Hierarchy Independence

The pupose of this subsection is the investigation of the undesired phenomenon, mention above, that the tautology property of a formula ϕ depends on symbols that do not occur in ϕ. It is totally unrelated to the rest of the book. Also its significance is unclear. It is a *loose end* that we saw no other way to publish. The average reader will just skip it. The curious reader may read on on his own risk and maybe finds some use for it.

Giese05

We use the concept of an abstract type from Giese [2005]. Abstract types are declared as a subset of the set of all types ATSym $\sqsubseteq$ TSym when a type hierarchy is fixed. In the definition of the semantic domain D we require for all elements $m \in D^A$ of an abstract type $A \in$ ATSym that there is a non-abstract type $B \in$ TSym $\setminus$ ATSym with $B \sqsubseteq A$ and $m \in D^B$.

For the purposes of this section we introduce the terminology.

def:Vocabulary

Definition 1.23. A *vocabulary* $\mathscr{V}$ is a pair $(\mathscr{T}, \Sigma)$ with $\mathscr{T} = (\mathrm{TSym}, \mathrm{ATSym}, \sqsubseteq)$ a type hierarchy and Σ a signature, such that every type A that occurs in the typing of a function or predicate symbol in Σ occurs in TSym, $A \in$ TSym.

def:VocExtension

Definition 1.24. A vocabulary $\mathscr{V}_2 = (\mathscr{T}_2, \Sigma_2)$ with $\mathscr{T}_2 = (\mathrm{TSym}_2, \mathrm{ATSym}_2, \sqsubseteq_2)$ is a simple extension of $\mathscr{V}_1 = (\mathscr{T}_1, \Sigma_1)$ with $\mathscr{T}_1 = (\mathrm{TSym}_1, \mathrm{ATSym}_1, \sqsubseteq_1)$, in symbols $\mathscr{V}_1 \leq_s \mathscr{V}_2$ if

- $\Sigma_1 \subseteq \Sigma_2$
- $\mathrm{TSym}_1 \subseteq \mathrm{TSym}_2$
- $\mathrm{ATSym}_2 \cap \mathrm{TSym}_1 = \mathrm{ATSym}_1$
- For every new $A \in \mathrm{TSym}_2$ (i.e. $A \notin \mathrm{TSym}_1$) there is a set $anc(A) \subseteq \mathrm{TSym}_1$, $\perp \notin anc(A)$ of types such that $\sqsubseteq_2$ on TSym_2 is the least partial order extending $\sqsubseteq_1$ that satiesfies $A \sqsubseteq_2 B$ for all $B \in anc(A)$.

$\mathscr{V}_2$ is called an extension of $\mathscr{V}_1$, in symbols $\mathscr{V}_1 \leq \mathscr{V}_2$ if there is a finite sequence of simple extensions $\mathscr{V}_1 \leq_s \mathscr{V}_1^0 \leq_s \ldots \leq_s \mathscr{V}_1^k \leq_s \mathscr{V}_2$.

We call $\mathscr{V}_1$ a *restriction* of $\mathscr{V}_2$ if $\mathscr{V}_2$ is an extension of $\mathscr{V}_1$.

lem:POExt

Lemma 1.25. *Let $(P_1, \leq_1)$ be a partial order, $P_1 \subseteq P_2$, and for every $p \in P_2 \setminus P_1$ there is a set $anc(p) \subseteq P_1$ and $\leq_2$ is the least partial order on P_2 that extends $\leq_1$ and satiesfies $p \leq_2 q$ for all $p \in P_2$ and $q \in anc(p)$. Then*

$$\leq_1 = \leq_2 \cap P_1 \times P_1$$

Proof. We will prove that $\leq_2$ can be explicitely defined by

$$p \leq_2 q \Leftrightarrow \begin{cases} p \leq_1 q & \text{if } p \in P_1, q \in P_1 \\ r \leq_1 q & \text{if } p \in (P_2 \setminus P_1), q \in P_1 \\ & \text{and } r \in anc(p) \\ p = \perp & \text{if } p \in P_1, q \in (P_2 \setminus P_1) \\ p = q & \text{if } p \in (P_2 \setminus P_1), q \in (P_2 \setminus P_1) \end{cases} \tag{1.1}$$

align:leastPOExt

The claim of the lemma follows immediately once we have established claim 1.1.

We need to show reflexivity, transitivity and antisymmetric of the relation $\leq_2$ thus defined.

Reflexivity follows directly from the cases $p,q \in P_1$ and $p,q \in P_2$.

Transitivity Consider $p \leq_2 q$, $q \leq_2 r$. The cases $p = q$ and $q = r$ are trivial. So we assume from now on $p \neq q$ and $q \neq r$.
Case $\mathbf{p} \in \mathbf{P_1}$
Subcase $\mathbf{q} \in (\mathbf{P_2} \setminus \mathbf{P_1})$
By definition we get $p = \bot$ and therefore $p = \bot \leq_2 r$ in any case.
Subcase $\mathbf{q} \in \mathbf{P_1}$ By definition this yields $p \leq_1 q$. Either $r \in P_1$ and $q \leq_1 r$, which by transitivity of $\leq_1$ also yields $p \leq_2 r$. Or $r \in (P_2 \setminus P_1)$ and $q = \bot$. But, then also $p = \bot$ and $p = \bot \leq_2 r$ follows.
Case $\mathbf{p} \in (\mathbf{P_2} \setminus \mathbf{P_1})$. $p \neq q$ implies $q \in P_1$ and $p' \leq_1 q$ for some $p' \in anc(p)$.
Subcase $\mathbf{r} \in \mathbf{P_1}$ We thus have $q \leq_1 r$, by transitivity of $\leq_1$ this yields $p' \leq_1 r$ and therefore $p \leq_2 r$.
Subcase $\mathbf{r} \in (\mathbf{P_2} \setminus \mathbf{P_1})$ In this case we must have $q = \bot$, which by $p' \leq_1 q$ also implies $p' = \bot$. But, this is ruled out by the definition of *anc*.

Antisymmetry Assume $q \leq_2 p$ and $p \leq_2 q$. If both p and q are in P_1 then $p = q$ follows from the antisymmetry of $\leq_1$. If both p and q are in $(P_2 \setminus P_1)$ then $p = q$ follows by definition of $\leq_2$. If $p \in P_1$ and $q \in (P_2 \setminus P_1)$ then we infer
$p = \bot$ from $p \leq_2 q$ and
$q' \leq_1 p = \bot$ for some $q' \in anc(q)$ from $q \leq_2 p$.
Thus $q' = \bot$, which is impossible. Thus this case cannot arrise. The remaining case $q \in P_1$ and $p \in (P_2 \setminus P_1)$ is handled symmetrically. □

cor:VocExtProp

Corollary 1.26. *If $\mathcal{V}_2 = (\mathcal{T}_2, \Sigma_2)$ with $\mathcal{T}_2 = (\mathrm{TSym}_2, \mathrm{ATSym}_2, \sqsubseteq_2)$ is an extension of $\mathcal{V}_1 = (\mathcal{T}_1, \Sigma_1)$ with $\mathcal{T}_1 = (\mathrm{TSym}_1, \mathrm{ATSym}_1, \sqsubseteq_1)$ then*

$$\sqsubseteq_1 = \sqsubseteq_2 \cap \mathrm{TSym}_1 \times \mathrm{TSym}_1$$

Proof. Follows by interated application of Lemma 1.25. □

Definition 1.24 is modelled after the extension of type hierachies in Java programs and is rather restrictive.

Example 1.27. If $\mathcal{V}_1 = (\mathcal{T}_1, \Sigma_1)$, $\mathcal{V}_2 = (\mathcal{T}_2, \Sigma_2)$ with $\Sigma_1 = \Sigma_2 = \emptyset$. $\mathrm{TSym}_1 = \{\bot, A, B, \top\}$, $\mathrm{ATSym}_1 = \emptyset$, and $\bot \sqsubseteq_1 A \sqsubseteq_1 \top$, $\bot \sqsubseteq_1 B \sqsubseteq_1 \top$. Furthermore $\mathrm{TSym}_2 = \{\bot, A, B, C, \top\}$, $\mathrm{ATSym}_2 = \emptyset$, and $\bot \sqsubseteq_2 A \sqsubseteq_2 \top$, $\bot \sqsubseteq_2 B \sqsubseteq_2 \top$ and $A \sqsubseteq_2 C \sqsubseteq_2 B$. Then $\mathcal{V}_2$ is not an extension of $\mathcal{V}_1$ since we have $(A,B) \in \sqsubseteq_2 \cap (\mathrm{TSym}_1 \times \mathrm{TSym}_1)$ but $(A,B) \notin \sqsubseteq_1$.

Let us modify this example a bit by changing the subtype relation of $\mathcal{T}_1$ to $\bot \sqsubseteq'_1 A \sqsubseteq'_1 B \sqsubseteq'_1 \top$. Still $\mathcal{V}_2$ is not an extension of $\mathcal{V}_1$. There is no way $A \sqsubseteq_2 C$ can be achieved for the new type C.

lem:HIndepSpecialCase

Lemma 1.28. *Let $\mathcal{V}_2 = (\mathcal{T}_2, \Sigma_2)$ with $\mathcal{T}_2 = (\mathrm{TSym}_2, \mathrm{ATSym}_2, \sqsubseteq_2)$ be an extension of $\mathcal{V}_1 = (\mathcal{T}_1, \Sigma_1)$ with $\mathcal{T}_1 = (\mathrm{TSym}_1, \mathrm{ATSym}_1, \sqsubseteq_1)$. Assume furthermore that for* *any* $C \in \mathrm{TSym}_2 \setminus \mathrm{TSym}_1$ *the intersection* $anc(C) \cap (\mathrm{TSym} \setminus \mathrm{ATSym}_1)$ *is a singleton then*

$$\vdash_{\mathscr{V}} \phi \quad \Leftrightarrow \quad \vdash_{\mathscr{V}_2} \phi$$

Proof.
By assumption we may write $anc(C) = \{anc_0(C)\}$ for every $C \in \mathrm{TSym}_2 \setminus \mathrm{TSym}_1$. Note, $anc_0(C) \notin aTypes_1$.
Part 1 Assume $\vdash_{\mathscr{V}} \phi$. Fix an arbitrary $\mathscr{V}_2$ structure $\mathscr{M}_2 = (M_2, \delta_2, I_2)$. We need to show $\mathscr{M}_2 \models \phi$. We define a $\mathscr{V}_1$ structure $\mathscr{M}_1 = (M_1, \delta_1, I_1)$ by

1. $M_1 = M_2$,
2. $\delta_1(d) = \begin{cases} \delta_2(d) & \text{if } \delta_2(d) \in \mathrm{TSym}_1 \\ anc_0(\delta_2(d)) & \text{otherwise} \end{cases}$
3. $I_1(f) = I_2(f)$, $I_1(p) = I_2(p)$ for $f, p \in \Sigma_1$.

Note, that the assumption on anc_0 make (2) a valid definition.

We observe for all $A \in \mathrm{TSym}_1$ that

$$\begin{aligned}
M_1^A &= \{d \in M_1 \mid \delta_1(d) \sqsubseteq_1 A\} \\
&= \{d \in M_2 \mid \delta_2(d) \sqsubseteq_1 A, \delta_2(d) \in \mathrm{TSym}_1\} \quad \cup \\
&\qquad \{d \in M_2 \mid anc_0(\delta_2(d)) \sqsubseteq_1 A, \delta_2(d) \notin \mathrm{TSym}_1\} \\
&= \{d \in M_2 \mid \delta_2(d) \sqsubseteq_2 A, \delta_2(d) \in \mathrm{TSym}_1\} \quad \cup \\
&\qquad \{d \in M_2 \mid \delta_2(d) \sqsubseteq_2 A, \delta_2(d) \notin \mathrm{TSym}_1\} \\
&= M_2^A
\end{aligned} \tag{1.2}$$

align:lem:HIndepSpecialCas

This shows that also the definitions $I_1(f) = I_2(f)$ and $I_1(p) = I_2(p)$ are correct, i.e., domain and range of $I_1(f)$ and $I_1(p)$ are as they should be.

Next we will show

$$\begin{gathered}
\text{For all variable assignments } \beta : VSym \to M_1 \text{ and all } \mathscr{V}_1 \text{ terms } t \\
\mathrm{val}_{\mathscr{M}_1,\beta}(t) = \mathrm{val}_{\mathscr{M}_2,\beta}(t
\end{gathered} \tag{1.3}$$

align:lem:HIndepSpecialCas

and

$$\begin{gathered}
\text{For all variable assignments } \beta : VSym \to M_1 \text{ and all } \mathscr{V}_1 \text{ formulas } \psi \\
\models_{\mathscr{M}_1,\beta} \psi \text{ iff } \models_{\mathscr{M}_2,\beta} \psi
\end{gathered} \tag{1.4}$$

align:lem:HIndepSpecialCas

Both, claims 1.3 and 1.4 can easily be proved by structural induction. The quantifier case in the proof of 1.4 uses 1.2.

By case assumption, $\vdash_{\mathscr{V}} \phi$, we know $\mathscr{M}_1 \models \phi$. Thus 1.4 yields $\mathscr{M}_2 \models \phi$, as desired.
Part 2 Assume $\vdash_{\mathscr{V}_2} \phi$. We fix an arbitrary $\mathscr{V}_1$ structure $\mathscr{M}_1 = (M_1, \delta_1, I_1)$ with the intention to show $\mathscr{M}_1 \models \phi$. The idea is to construct a $\mathscr{V}_2$ structure $\mathscr{M}_2$ such that $\mathscr{M}_1$ is obtained from $\mathscr{M}_2$ by the construction described in Part 1. We leave the details to the reader. The rest of the proof then follows as in Part 1. □

The examples and Lemma 1.4 show that the perceived problem has to do with abstract types. The next lemma investigates an approach to get along without abstract types.

lem:NoAbstractTypes

Lemma 1.29. *Let $\mathscr{V} = (\mathscr{T}, \Sigma)$ be a vocabulary with $\mathscr{T} = (\mathrm{TSym}, \mathrm{ATSym}, \sqsubseteq)$. Let $\mathscr{V}_1$ by another vocabulary that differs from $\mathscr{V}$ only by the fact that no type is abstract, i.e. $\mathscr{V}_1 = (\mathscr{T}_1, \Sigma_1)$ with $\mathscr{T}_1 = (\mathrm{TSym}_1, \mathrm{ATSym}_1, \sqsubseteq_1)$, such that $\Sigma_1 = \Sigma$, $\mathrm{TSym}_1 = \mathrm{TSym}$, $A \sqsubseteq_1 B \Leftrightarrow A \sqsubseteq B$ and $\mathrm{ATSym}_1 = \emptyset$.*

For any type $A^i \in \mathrm{ATSym}$ let $B^i_1, \ldots, B^i_{n_i}$ be all non-abstract subtypes of A^i, for $0 \leq i < k$.

Let F^i denote the variablefree formula $\forall A^i\ x^i. \bigvee_{j=0}^{n_i} \exists B^i_j\ y_j. x^i \doteq y_j$ and $F = \bigwedge_{i=0}^{k} F^i$.

A $\mathscr{V}_1$ structure (M, δ, I) is a $\mathscr{V}$ structure if and only if $\mathscr{V}_1 \models F$

Proof. If (M, δ, I) is a $\mathscr{V}$ structure then it is also a $\mathscr{V}_1$ structure. Since the range of δ is disjoint from ATSym we see that $\mathscr{V}_1 \models F^i$ for all $0 \leq i < k$.

If, on the other hand, (M, δ, I) is a $\mathscr{V}_1$ structure we could have $\delta(d) = A^i$ for some $d \in M$ and $A^i \in \mathrm{ATSym}$ which would prevent it from being a $\mathscr{V}$ structure. We will argue that because of $(M, \delta, I) \models F^i$ this cannot happen. From $(M, \delta, I) \models F^i$ we read off a non-abstract subtype B^i_j of A^i and an element $e \in M^{B^i_j}$ with $d = e$. By definition of $M^{B^i_j}$ we get $\delta(e) \sqsubseteq B^i_j \sqsubset A^i$ and thus $\delta(d) = \delta(e) \neq A^i$. □

cor:NoAbstractTypes

Corollary 1.30. *With the notation from Lemma 1.29 and any $\mathscr{V}$ formula ϕ we have*

$$\vdash_{\mathscr{V}} \phi \Leftrightarrow F \vdash_{\mathscr{V}_1} \phi$$

Proof. Follows easily from Lemma 1.29.

End of the Digression on Hierarchy Independence

1.2.5 Digression on Completeness Proof

subsect:CompletenessProof

The proof of Theorem 1.21 proceeds by three lemmata. Lemma 1.33 is devoted to the proof of soundness and Lemma 1.34 covers completeness. Lemma 1.31 is need as a preparatory step for both.

lemma:soundCompleteRules

Lemma 1.31. *All rules in Figures 1.1 and 1.2 are sound and complete*

Proof of Lemma 1.31

We list all rules and the proof obligation for their soundness and completeness. The proofs are either trivial or totally trivial. In the cases impLeft, allRight, allLeft, exLeft, and eqDynamicSort we include the details. The correctness proof of the last rule is particularly demanding.

andLeft $\dfrac{\Gamma, \phi, \psi \Longrightarrow \Delta}{\Gamma, \phi \wedge \psi \Longrightarrow \Delta}$

proof obligation:

$\bigwedge\Gamma \land \phi \land \psi \to \bigvee\Delta$ is universally valid
iff
$\bigwedge\Gamma \land (\phi \land \psi) \to \bigvee\Delta$ is universally valid

$$\mathsf{andRight}\ \frac{\Gamma \Longrightarrow \phi, \Delta \qquad \Gamma \Longrightarrow \psi, \Delta}{\Gamma \Longrightarrow \phi \land \psi, \Delta}$$

proof obligation:

$(\bigwedge\Gamma \to \bigvee\Delta \lor \phi)$ and $(\bigwedge\Gamma \to \bigvee\Delta \lor \psi)$ are universally valid
iff
$\bigwedge\Gamma \to \bigvee\Delta \lor (\phi \land \psi)$ is universally valid

$$\mathsf{orRight}\ \frac{\Gamma \Longrightarrow \phi, \psi, \Delta}{\Gamma \Longrightarrow \phi \lor \psi, \Delta}$$

proof obligation:

$(\bigwedge\Gamma \to \bigvee\Delta \lor \phi \lor \psi)$ is universally valid
iff
$(\bigwedge\Gamma \to \bigvee\Delta \lor (\phi \lor \psi))$ is universally valid

$$\mathsf{orLeft}\ \frac{\Gamma, \phi \Longrightarrow \Delta \qquad \Gamma, \psi \Longrightarrow \Delta}{\Gamma, \phi \lor \psi \Longrightarrow \Delta}$$

proof obligation:

$(\bigwedge\Gamma \land \phi \to \bigvee\Delta)$ and $(\bigwedge\Gamma \land \psi \to \bigvee\Delta)$ are universally valid
iff
$(\bigwedge\Gamma \land (\phi \lor \psi) \to \bigvee\Delta)$ is universally valid

$$\mathsf{impRight}\ \frac{\Gamma, \phi \Longrightarrow \psi, \Delta}{\Gamma \Longrightarrow \phi \to \psi, \Delta}$$

proof obligation:

$(\bigwedge\Gamma \land \phi \to \bigvee\Delta \lor \psi)$ is universally valid
iff
$(\bigwedge\Gamma \to \bigvee\Delta \lor (\phi \to \psi))$ is universally valid

$$\mathsf{impLeft}\ \frac{\Gamma \Longrightarrow \phi, \Delta \qquad \Gamma, \psi \Longrightarrow \Delta}{\Gamma, \phi \to \psi \Longrightarrow \Delta}$$

proof obligation:

$(\bigwedge\Gamma \to \bigvee\Delta \lor \phi)$ and $(\bigwedge\Gamma \land \psi \to \bigvee\Delta)$ are universally valid
iff
$(\bigwedge\Gamma \land (\phi \to \psi) \to \bigvee\Delta)$ is universally valid

Proof. $\Downarrow$:
Assume that $\bigwedge\Gamma \to \bigvee\Delta \lor \phi$ and $\bigwedge\Gamma \land \psi \to \bigvee\Delta$ are both universally valid. To

prove that $\bigwedge\Gamma \wedge (\phi \rightarrow \psi) \rightarrow \bigvee\Delta$ is universally valid we fix an arbitrary first-order structure $\mathscr{M}$ for a type hierarchy $\mathscr{T}$ and signature Σ such that the three formulas under consideration are contained in $Fml_{\mathscr{T},\Sigma}$. We assume $\mathscr{M} \models \bigwedge\Gamma \wedge (\phi \rightarrow \psi)$ and need to show $\mathscr{M} \models \bigvee\Delta$. In particular, we have $\mathscr{M} \models \phi \rightarrow \psi$ at our disposal. There are two cases to be considered. In the frist we have $\mathscr{M} \models \neg\phi$. By validity of $\bigwedge\Gamma \rightarrow \bigvee\Delta \vee \phi$ we get $\mathscr{M} \models \bigvee\Delta$ and are finished. In the second case we have $\mathscr{M} \models \psi$. Now, we obtain $\mathscr{M} \models \bigvee\Delta$ from the assumed validity of $\bigwedge\Gamma \wedge \psi \rightarrow \bigvee\Delta$.
$\Uparrow$:
We assume that $\bigwedge\Gamma \wedge (\phi \rightarrow \psi) \rightarrow \bigvee\Delta$ is universally valid and fix an arbitrary first-order structure $\mathscr{M}$ for a type hierarchy $\mathscr{T}$ and signature Σ such that the three formulas under consideration are contained in $Fml_{\mathscr{T},\Sigma}$. If $\mathscr{M} \models \neg\bigwedge\Gamma$ then $\mathscr{M}$ trivially satisfies the two formulas in the succedent of the rule. So we assume from now on $\mathscr{M} \models \bigwedge\Gamma$.Now, if $\mathscr{M} \models \phi$ then we first obtain obviously $\mathscr{M} \models \bigvee\Delta \vee \phi$, as desired. We still need to show $\mathscr{M} \models \bigwedge\Gamma \wedge \psi \rightarrow \bigvee\Delta$. So we assume $\mathscr{M} \models \psi$. We are now in the situation that $\mathscr{M} \models \bigwedge\Gamma \wedge (\phi \rightarrow \psi)$. Since $\bigwedge\Gamma \wedge (\phi \rightarrow \psi) \rightarrow \bigvee\Delta$ is universally valid this entails $\mathscr{M} \models \bigvee\Delta$ and we are finished in this case.
It remains to deal with $\mathscr{M} \models \neg\phi$. Since we have $\mathscr{M} \models \bigwedge\Gamma \wedge (\phi \rightarrow \psi)$ in this case, the assumption gives us immediately $\mathscr{M} \models \bigvee\Delta$. From the we see that again $\mathscr{M}$ satisfies both formuluas in the conclusion. □

$$\mathsf{notLeft}\ \frac{\Gamma \Longrightarrow \phi, \Delta}{\Gamma, \neg\phi \Longrightarrow \Delta}$$

proof obligation:

$(\bigwedge\Gamma \rightarrow \bigvee\Delta \vee \phi)$ is universally valid
iff
$(\bigwedge\Gamma \wedge \neg\phi \rightarrow \bigvee\Delta)$ is universally valid

$$\mathsf{notRight}\ \frac{\Gamma, \phi \Longrightarrow \Delta}{\Gamma \Longrightarrow \neg\phi, \Delta}$$

proof obligation:

$(\bigwedge\Gamma \wedge \phi \rightarrow \bigvee\Delta)$ is universally valid
iff
$(\bigwedge\Gamma \rightarrow \bigvee\Delta \vee \neg\phi)$ is universally valid

$$\mathsf{allRight}\ \frac{\Gamma \Longrightarrow [x/c](\phi), \Delta}{\Gamma \Longrightarrow \forall x.\phi, \Delta} \quad \text{with } c : \rightarrow A \text{ a new constant, if } x{:}A$$

proof obligation:

$(\bigwedge\Gamma \rightarrow \bigvee\Delta \vee [x/c](\phi))$ is universally valid
iff
$(\bigwedge\Gamma \rightarrow \bigvee\Delta \vee \forall x.\phi)$ is universally valid

Proof. We assume first that $(\bigwedge\Gamma \rightarrow \bigvee\Delta \vee \forall x.\phi)$ is universally valid and consider an arbitrary structure $\mathscr{M}$ satisfying $\mathscr{M} \models \bigwedge\Gamma$ with the aim of showing $\mathscr{M} \models \bigvee\Delta \vee$

$[x/c](\phi)$. By assumption we get immediately $\mathscr{M} \models \bigvee\Delta \vee \forall x.\phi)$. If $\mathscr{M} \models \bigvee\Delta$ is true we are done. If on the other hand $\mathscr{M} \models \vee\forall x.\phi)$ is the case we also get $\mathscr{M} \models [x/c](\phi)$.

Now, assume that $(\bigwedge\Gamma \to \bigvee\Delta \vee [x/c](\phi))$ is universally valid and consider an arbitrary structure $\mathscr{M}$ satisfying $\mathscr{M} \models \bigwedge\Gamma$ with the aim of showing $\mathscr{M} \models \bigvee\Delta \vee \forall x.\phi)$. If $\mathscr{M} \models \bigvee\Delta$ is true we are again done. It remains to consider the case $\mathscr{M} \models \neg\bigvee\Delta$. Let $\mathscr{M}'$ be a structure that differs from $\mathscr{M}$ only in the interpetation of the new constant c. Since c is new we still have $\mathscr{M}' \models \bigwedge\Gamma$ and $\mathscr{M}' \models \neg\bigvee\Delta$. By the assumed universall validity this implies $\mathscr{M}' \models [x/c](\phi)$. This argument suffices to show $\mathscr{M} \models \forall x.\phi)$. □

$$\mathsf{allLeft}\ \frac{\Gamma, \forall x.\phi, [x/t](\phi) \Longrightarrow \Delta}{\Gamma, \forall x.\phi \Longrightarrow \Delta} \quad \text{with } t \in \mathrm{Trm}_{A'} \text{ ground, } A' \sqsubseteq A, \text{ if } x{:}A$$

proof obligation:

$(\bigwedge\Gamma \wedge \forall x.\phi \wedge [x/t](\phi) \to \bigvee\Delta)$ is universally valid
iff
$(\bigwedge\Gamma \wedge \forall x.\phi \to \bigvee\Delta)$ is universally valid

Proof. Assume that $(\bigwedge\Gamma \wedge \forall x.\phi \wedge [x/t](\phi) \to \bigvee\Delta)$ is universally valid and consider an arbitrary structure $\mathscr{M}$ satisfying $\mathscr{M} \models \bigwedge\Gamma \wedge \forall x.\phi$ with the aim of showing $\mathscr{M} \models \bigvee\Delta$. Since $\mathscr{M} \models \forall x.\phi$ implies $\mathscr{M} \models [x/t](\phi)$ we can make use of the assumption to obtain $\mathscr{M} \models \bigvee\Delta$ as desired.

The reverse implication is trivial. □

$$\mathsf{exLeft}\ \frac{\Gamma, [x/c](\phi) \Longrightarrow \Delta}{\Gamma, \exists x.\phi \Longrightarrow \Delta} \text{ with } c : \to A \text{ a new constant, and } x{:}A.$$

proof obligation:

$\bigwedge\Gamma \wedge [x/c](\phi) \to \bigvee\Delta$ is universally valid
iff
$\bigwedge\Gamma \wedge \exists x.\phi \to \bigvee\Delta$ is universally valid

Proof. Assume $\bigwedge\Gamma \wedge \exists x.\phi \to \bigvee\Delta$ is universally valid and consider an arbitrary structure $\mathscr{M}$ satisfying $\mathscr{M} \models \Gamma \wedge [x/c](\phi)$ with the aim of showing $\mathscr{M} \models \bigvee\Delta$. Since $\mathscr{M} \models [x/c](\phi)$ entails $\mathscr{M} \models \exists x.\phi$ we may use the assumption to conclude $\mathscr{M} \models \bigvee\Delta$ as desired.

Now assume that $\bigwedge\Gamma \wedge [x/c](\phi) \to \bigvee\Delta$ is universally valid and consider an arbitrary structure $\mathscr{M}$ satisfying $\mathscr{M} \models \bigwedge\Gamma \wedge \exists x.\phi$ with the aim of showing $\mathscr{M} \models \bigvee\Delta$. Let $a \in M$ be an element that serves as a witness for the existential quantifier. Let $\mathscr{M}'$ that differs from $\mathscr{M}$ only in the interpretation of the new constant c which we define in $\mathscr{M}'$ to be a. Since c is a new constant we still have $\mathscr{M}' \models \bigwedge\Gamma$. Since $\mathscr{M}' \models [x/c](\phi)$ is true, we obtain from the assumption $\mathscr{M}' \models \bigvee\Delta$. Since c was a new constant this implies also $\mathscr{M} \models \bigvee\Delta$, as desired. □

exRight $\dfrac{\Gamma \Longrightarrow \exists x.\phi, [x/t](\phi), \Delta}{\Gamma \Longrightarrow \exists x.\phi, \Delta}$ with $t \in \mathrm{Trm}_{A'}$ ground, $A' \sqsubseteq A$, and $x{:}A$.

proof obligation:

$\bigwedge \Gamma \to \bigvee \Delta \vee \exists x.\phi \vee [x/t](\phi)$ is universally valid
iff
$\bigwedge \Gamma \to \bigvee \Delta \vee \exists x.\phi$ is universally valid

close $\dfrac{*}{\Gamma, \phi \Longrightarrow \phi, \Delta}$

proof obligation:

$\bigwedge \Gamma \wedge \phi \to \bigvee \Delta \vee \phi$ is universally valid

closeFalse $\dfrac{*}{\Gamma, \mathrm{false} \Longrightarrow \Delta}$

proof obligation:

$\bigwedge \Gamma \wedge \mathrm{false} \to \bigvee \Delta$ is universally valid

closeTrue $\dfrac{*}{\Gamma \Longrightarrow \mathrm{true}, \Delta}$

proof obligation:

$\bigwedge \Gamma \to \bigvee \Delta \vee \mathrm{true}$ is universally valid

eqLeft $\dfrac{\Gamma, t_1 \doteq t_2, [z/t_1](\phi), [z/t_2](\phi) \Longrightarrow \Delta}{\Gamma, t_1 \doteq t_2, [z/t_1](\phi) \Longrightarrow \Delta}$ if $\sigma(t_2) \sqsubseteq \sigma(t_1)$

proof obligation:

$\bigwedge \Gamma \wedge t_1 \doteq t_2 \wedge [z/t_1](\phi) \wedge [z/t_2](\phi) \to \bigvee \Delta$ is universally valid
iff
$\bigwedge \Gamma \wedge t_1 \doteq t_2 \wedge [z/t_1](\phi) \to \bigvee \Delta$ is universally valid

eqRight $\dfrac{\Gamma, t_1 \doteq t_2 \Longrightarrow [z/t_2](\phi), [z/t_1](\phi), \Delta}{\Gamma, t_1 \doteq t_2 \Longrightarrow [z/t_1](\phi), \Delta}$ if $\sigma(t_2) \sqsubseteq \sigma(t_1)$

proof obligation:

$(\bigwedge \Gamma \wedge t_1 \doteq t_2 \to \bigvee \Delta \vee [z/t_1](\phi) \vee [z/t_2](\phi))$ is universally valid
iff
$(\bigwedge \Gamma \wedge t_1 \doteq t_2 \to \bigvee \Delta \vee [z/t_1](\phi))$ is universally valid

eqSymmLeft $\dfrac{\Gamma, t_2 \doteq t_1 \Longrightarrow \Delta}{\Gamma, t_1 \doteq t_2 \Longrightarrow \Delta}$

proof obligation:

$\bigwedge\Gamma \wedge t_2 \doteq t_1 \rightarrow \bigvee\Delta$ is universally valid
iff
$\bigwedge\Gamma \wedge t_1 \doteq t_2 \rightarrow \bigvee\Delta$ is universally valid

$$\mathsf{eqClose}\ \frac{*}{\Gamma \Longrightarrow t \doteq t, \Delta}$$

proof obligation:

$\bigwedge\Gamma \rightarrow \bigvee\Delta \vee t \doteq t$ is universally valid

$$\mathsf{eqDynamicSort}\ \frac{\Gamma, t_1 \doteq t_2, \exists x(x \doteq t_1 \wedge x \doteq t_2) \Longrightarrow \Delta}{\Gamma, t_1 \doteq t_2 \Longrightarrow \Delta}$$

if $\sigma(t_1)$ and $\sigma(t_2)$ are incomparable,
the sort C of x is new and satisfies
$C \sqsubset \sigma(t_1)$ and $C \sqsubset \sigma(t_2)$

Proof. To prove soundness of **eqDynamicSort** we assume that $\bigwedge\Gamma \wedge t_1 \doteq t_2 \wedge \exists x.(x \doteq t_1 \wedge x \doteq t_2) \rightarrow \bigvee\Delta$ is universally valid for type hierarchy $\mathscr{T}_C$ and need to show that $\bigwedge\Gamma \wedge t_1 \doteq t_2 \rightarrow \bigvee\Delta$ is is universally valid for the hierarchy $\mathscr{T}$.

The type hierarchy $\mathscr{T}_C = (\text{TSym} \cup \{C\}, \sqsubseteq_C)$ is an extension of the hierarchy $\mathscr{T} = (\text{TSym}, \sqsubseteq)$ in the sense of Definition 1.18 , with $\sqsubseteq_C$ the least subtype relation containing $\sqsubseteq \cup \{(C, \sigma(t_1)), (C, \sigma(t_2))\}$.

Assume that $\bigwedge\Gamma \wedge t_1 \doteq t_2 \wedge \exists x.(x \doteq t_1 \wedge x \doteq t_2) \rightarrow \bigvee\Delta$ is universally valid for type hierarchy $\mathscr{T}_C$. To prove universal validity of $\bigwedge\Gamma \wedge t_1 \doteq t_2 \rightarrow \bigvee\Delta$ we need to consider a type hierarchy $\mathscr{T}^1 = (\text{TSym}^1, \sqsubseteq^1)$ that is an arbitrary extension of $\mathscr{T} = (\text{TSym}, \sqsubseteq)$ and an arbitrary $(\mathscr{T}^1, \Sigma)$-structure $\mathscr{M} = (M, \delta, I)$ satisfying $\mathscr{M} \models \Gamma \wedge t_1 \doteq t_2$ with the aim of showing $\mathscr{M} \models \bigvee\Delta$. Let T be the dynamic type of $t_1^{\mathscr{M}} = t_2^{\mathscr{M}}$, i.e., $\delta(t_1^{\mathscr{M}}) = \delta(t_2^{\mathscr{M}}) = T$, which obviously satisfies $T \sqsubseteq \sigma(t_1)$ and $T \sqsubseteq \sigma(t_2)$. Set $\mathscr{T}_C^1 = (\text{TSym}^1 \cup \{C\}, \sqsubseteq_C^1)$ with $\sqsubseteq_C^1$ be the smallest subtype relation containing $\sqsubseteq^1 \cup \{(C, \sigma(t_1)), (C, \sigma(t_2))\}$. We need a further extension $\mathscr{T}^2 = (\text{TSym}^1 \cup \{C, C^2\}, \sqsubseteq^2)$ of $\mathscr{T}_C^1$, where $\sqsubseteq^2$ is the smallest subtype relation containing $\sqsubseteq_C^1 \cup \Delta$ with $\Delta = \{(C^2, C), (C^2, T)\}$.

We proceed with main proof by constructing a $(\mathscr{T}^2, \Sigma)$-structure $\mathscr{M}^2 = (M, \delta^2, I)$ that differs from $\mathscr{M}$ only in δ^2 which is given by

$$\delta^2(o) = \begin{cases} C^2 & \text{if } o = t_1^{\mathscr{M}} = t_2^{\mathscr{M}} \\ \delta(o) & \text{otherwise .} \end{cases}$$

This leads to $\mathscr{M}^2 \models \exists x.(x \doteq t_1 \wedge x \doteq t_2)$ – if we remember that x is a variable of the type C and $C^2 \sqsubseteq C$.

The crucial property of the type hierarchy $\mathscr{T}^2$ is

$$\text{for any } o \in M \text{ and any } A \in \text{TSym}^1 \quad \delta(o) \sqsubseteq^1 A \Leftrightarrow \delta^2(o) \sqsubseteq^2 A\ . \qquad (1.5)$$ align:C2ext

Here are the arguments why (1.5) is true: In case $o \neq t_1^{\mathscr{M}}$ we have $\delta^2(o) = \delta(o) \in \text{TSym}^1$ and $\delta(o) \sqsubseteq^1 A \Leftrightarrow \delta(o) \sqsubseteq^2 A$ by item 1 of Lemma 1.19. In case $o = t_1^{\mathscr{M}} = t_2^{\mathscr{M}}$

we have $\delta(o) = T$ and $\delta^2(o) = C^2$. By item 2 of Lemma 1.19, $C^2 \sqsubseteq^2 A$ is equivalent to the disjunction of $T \sqsubseteq^1_C A$ or $C \sqsubseteq^1_C A$. Again by Lemma 1.19, this is equivalent to $T \sqsubseteq^1 A$ or $\sigma(t_1) \sqsubseteq^1 A$ or $\sigma(t_2) \sqsubseteq^1 A$. Since $\sigma(t_1), \sigma(t_2) \sqsubseteq T$, this is equivalent to $T \sqsubseteq^1 A$. In total, we have shown (1.5) by proving that $C^2 \sqsubseteq^2 A$ is equivalent to $T \sqsubseteq^1 A$.

We need to convince ourselves that $\mathscr{M}^2 \models \Gamma \wedge t_1 \doteq t_2$ is still true. We will prove the following auxiliary statement:

$$\begin{array}{c}\text{Let } \phi \text{ be an arbitrary } (\mathscr{T},\Sigma)\text{-formula, } \beta \text{ a variable assignment, then}\\ (\mathscr{M},\beta) \models \phi \quad \Leftrightarrow \quad (\mathscr{M}^2,\beta) \models \phi\end{array} \tag{1.6}$$

align:Cext

The proof of (1.6) proceeds by induction on the complexity of ϕ. The only non-trivial steps are the induction steps for quantifiers. So assume that the claim is true for $\phi(x_1,\ldots,x_n)$[1] and we try to establish it for $(\exists x_1.\phi)(x_2,\ldots,x_n)$, with x_1 a variable of type A. By choice of ϕ, the type A is different from C and C^2.

$$\begin{array}{rll}(\mathscr{M},\beta) \models \exists x_1.\phi & \Leftrightarrow (\mathscr{M},\beta^o_{x_1}) \models \phi(x_1) & \text{for } o \in M \text{ with } \delta(o) \sqsubseteq A\\ & \Leftrightarrow (\mathscr{M}^2,\beta^o_{x_1}) \models \phi(x_1) & \text{induction hypothesis}\\ & \Leftrightarrow (\mathscr{M}^2,\beta) \models \exists x_1.\phi & \text{by } (1.5)\end{array}$$

Now, we have established $\mathscr{M}^2 \models \Gamma \wedge t_1 \doteq t_2$. From the assumption we obtain $\mathscr{M}^2 \models \bigvee\Delta$, which entails $\mathscr{M} \models \bigvee\Delta$ by another appeal to (1.6), as desired. □

$$\text{eqRef}\ \frac{\Gamma, t \doteq t \Longrightarrow \Delta}{\Gamma \Longrightarrow \Delta}$$

proof obligation:

$$\begin{array}{l}\bigwedge\Gamma \wedge t \doteq t \to \bigvee\Delta \text{ is universally valid}\\ \text{iff}\\ \bigwedge\Gamma \to \bigvee\Delta \text{ is universally valid}\end{array}$$

For the soundness proof we need the following auxiliary definition

defi:ProofTreeHeight

Definition 1.32. Let T be a finite proof tree. For every node n in T we define its height $h(n)$ recursively by

1. $h(n) = 0$ if n is a leaf.
2. $h(n) = max\{h(n_1), h(n_2)\} + 1$ if n_1, n_2 are the children of n. If there is only one child n_1 of n this reduces, of course, to $h(n) = h(n_1) + 1$.

lemma:FOLSoundness

Lemma 1.33. *If T is a closed proof tree and ϕ a label of a node in T, then ϕ is universally valid.*

[1] i.e., a formula with at most the free variables $x_1,\ldots,x_n$

Proof of Lemma 1.33

Let S be the sequent that labels node n in T. The lemma is proved by induction on $h(n)$. If $h(n) = 0$ then S is the premise of a closing rule and by Lemma 1.31 S is universally valid. If $h(n) > 0$ and n_1, n_2 are the children of n, then $h(n_i) < h(n)$. So we know by the indcution hypothesis that the sequents S_1, S_2 labeling n_1 and n_2 are universally valid. Now, Lemma 1.31 yields universal validity of S. □

lemma:FOLCompleteness

Lemma 1.34. *Let $S = \Gamma \Longrightarrow \Delta$ be a sequent with $\Gamma, \Delta \subset \mathrm{Fml}_{\mathcal{T},\Sigma}$ for some type hierachy $\mathcal{T}$ and signature Σ.*
If S is universally valid then there is a closed proof tree with root labeled by S.

Proof of Lemma 1.34

The proof proceeds by contradiction. Assume there is no closed proof tree with root labeled by S. Let T be a proof tree with root labeled by S such that all rules have been exhaustively applied, but T is not closed. Because of the rules allLeft and exRight T is necessarily infinite. By an appeal to König's Lemma there is an infinite branch B of T that is not closed.

Let H_0 be the set of all ground terms. We define the relation $\sim_B$ on H_0 by

$$t_1 \sim_B t_2 \text{ iff } t_1 = t_2 \text{ or}$$
$$\text{there is a sequent } \Gamma \Longrightarrow \Delta \text{ in } B \text{ with } t_1 \doteq t_2 \in \Gamma$$

The relation $\sim_B$ is an equivalence relation. Reflexivity is assured by definition. If $t_1 \sim_B t_2$ with $t_1 \doteq t_2 \in \Gamma$ and $\Gamma \Longrightarrow \Delta \in B$, then somewhere in B the rule eqSymmLeft must have been applied since we assume exhaustive rule application. Thus there will be a sequent $\Gamma' \Longrightarrow \Delta' \in B$ with $t_2 \doteq t_1 \in \Gamma'$ and we arrive at $t_2 \sim_B t_1$. It remains to show transitivity. We start from $t_1 \sim_B t_2$ and $t_2 \sim_B t_3$. By definition of $\sim_B$ there are sequents $\Gamma_1 \Longrightarrow \Delta_1$ and $\Gamma_2 \Longrightarrow \Delta_2$ in B with $t_1 \doteq t_2 \in \Gamma_1$ and $t_2 \doteq t_3 \in \Gamma_2$. Since there is no rule that drops an equality in the antecedent, only the arguments may be swaped, there will be a sequent $\Gamma' \Longrightarrow \Delta'$ in B such that $t_1 \doteq t_2$ or $t_2 \doteq t_1$ and at the same time $t_2 \doteq t_3$ or $t_3 \doteq t_2$ occur in Γ'. We consider each case separately.

1. $t_1 \doteq t_2$ and $t_2 \doteq t_3$
2. $t_1 \doteq t_2$ and $t_3 \doteq t_2$
3. $t_2 \doteq t_1$ and $t_2 \doteq t_3$
4. $t_2 \doteq t_1$ and $t_3 \doteq t_2$

In case (1) we use eqLeft to replace the lefthand side t_2 of the second equation by its righthand side in the first equation and obtain $t_1 \doteq t_3$.

In case (3) we replace, using eqLeft, the lefthand side t_2 of the first equation by its righthand side in the second equation and obtain $t_1 \doteq t_3$.

In case (4) replace the lefthand side t_2 of the first equation by its righthand side in the second equation and obtain $t_3 \doteq t_1$. Another application of eqSymmLeft yields $t_1 \doteq t_3$.

Case (2) is the most involved. By exhaustiveness of B we know that eqSymmLeft will be applied again to both equations in focus. If it is first applied to the first equation we obtain the situation in case (4). If it is first applied to the second equation we obtain case (1).

Thus in any case $t_1 \sim_B t_3$ follows and transitivity is established. In total we know now that $\sim_B$ is an equivalence relation.

Next we aim to show that $\sim_B$ is also a congruence relation, i.e., $t_i \sim_B t'_i$ for $1 \leq i \leq n$ implies $f(t_1,\ldots,t_n) \sim_B f(t'_1,\ldots,t'_n)$ and $q(t_1,\ldots,t_n) \leftrightarrow q(t'_1,\ldots,t'_n)$ for any n-place function symbol f and n-place predicate symbol q. For simplicity we only present that case of a unary function symbol f. We need to show $f(t) \sim_B f(t')$ from $t \sim_B t'$. We first consider the case that $\sigma(t)$ and $\sigma(t')$ are comparable, e.g., $\sigma(t') \sqsubseteq \sigma(t)$. By assumption there is a sequent $\Gamma \Longrightarrow \Delta$ in branch B with $t \doteq t' \in \Gamma$. From the argument given above, we know that there is also sequent $\Gamma_1 \Longrightarrow \Delta_1$ on B with $f(t) \doteq f(t) \in \Gamma_1$ and $t \doteq t' \in \Gamma_1$. By rule **eqLeft** we obtain a sequent $\Gamma_2 \Longrightarrow \Delta_2$ on B with $f(t) \doteq f(t') \in \Gamma_2$, and thus $f(t) \sim_B f(t')$. It remains to deal with the case that $\sigma(t)$ and $\sigma(t')$ are incomparable. Then rule **eqDynmicSort** applies and yields a sequent $\Gamma_3 \Longrightarrow \Delta_3$ on B with $t \doteq t' \in \Gamma_3$ and also $\exists x(x \doteq t \land x \doteq t') \in \Gamma_3$ with x a variable of the new type C, with $C \sqsubseteq \sigma(t)$ and $C \sqsubseteq \sigma(t')$. By **exLeft** there is a Skolem symbol sk of type C with $sk \sim_B t$ and $sk \sim_B t'$. By the comparable types case of the congruence property already established we obtain $f(sk) \sim_B f(t)$ and $f(sk) \sim_B f(t')$ in total $f(t) \sim_B f(t')$ as desired. The case of arbitrary n-place function symbols is only marginally more complicated. The congruence property for predicate symbols follows along the same line, starting from the fact that there is a sequent $\Gamma \Longrightarrow \Delta$ on B with $(p(t) \leftrightarrow p(t')) \in \Gamma$.

By $[t]_B$ for $t \in H_0$ we denote the equivalence of t with respect to $\sim_B$, i.e., $[t]_B = \{s \in H_0 \mid t \sim_B s\}$.

$$H = \{[t]_B \mid t \in H_0\}$$

Next we need to decide what (dynamic) type an element $[t]_B$ should have in the structure to be constructed. We call an equivalence class $[t]_B$ *typed* if there is a type $T_0 \in \mathscr{T}$ such that there is an term $t_0 \in [t]_B$ with $\sigma(t_0) = T_0$ and for all $t' \in [t]_B$ the subtype relation $T_0 \sqsubseteq \sigma(t')$ holds true. For every equivalence class $[t]_B$ that is not typed we introduce a new type constant $T_{[t]}$ and set

$$\begin{aligned} \Delta &= \{T_{[t]} \mid [t]_B \in H \text{ is not typed}\} \\ \Delta_R &= \{T_{[t]} \sqsubseteq \sigma(t') \mid t' \in [t]_B \text{ and } T_{[t]} \in \Delta\} \end{aligned}$$

The type hierarchy $\mathscr{T}' = (\text{TSym}', \sqsubseteq')$ extending $\mathscr{T} = (\text{TSym}, \sqsubseteq)$ is given by

$$\begin{aligned} &\text{TSym}' = \text{TSym} \cup \Delta \\ &\sqsubseteq' \quad\;\; = \text{transitive closure of } \sqsubseteq \cup \Delta_R \end{aligned}$$

Obviously, $\mathscr{T} \sqsubseteq \mathscr{T}'$.

We are now ready to define a first-order $(\mathscr{T}', \Sigma)$-structure $\mathscr{M} = (H, \delta, I)$.

The (dynamic) typing function is given by

$$\delta([t]_B) = \begin{cases} T_0 & \text{if } [t]_B \text{ is typed by } T_0 \\ T_{[t]} & \text{the new type constant, otherwise} \end{cases}$$

For any n-place function symbol f we set

$$I(f)([t_1]_B, \dots, [t_n]_B) = [f(t_1, \dots, t_n)]_B$$

For this to be an unambiguous definition we need to show (we only present the case $n = 1$, which can easily be generalized) that $t \sim_B s$ implies $f(t) \sim_B f(s)$. If $t \sim_B s$ then there is a sequent $\Gamma, t \doteq s \Longrightarrow \Delta$ in B. By eqRef we also find a sequent $\Gamma, t \doteq s, f(t) \doteq f(t) \Longrightarrow \Delta$ in B. Using rule eqLeft we see that a sequent $\Gamma', f(s) \doteq f(t) \Longrightarrow \Delta'$ occurs in B. This shows $f(t) \sim_B f(s)$.

For any n-place predicate symbol p we set

$$I(p) = \{([t_1]_B, \dots, [t_n]_B) \mid \text{ a sequent } \Gamma, p([t_1, \dots, t_n) \Longrightarrow \Delta \text{ occurs in } B\}$$

Again we have to argue that this definition is unambiguous. We do this again for the special case $n = 1$. The generalization to arbitrary n is left as an easy exercise to the reader. If $\Gamma, p(t) \Longrightarrow \Delta$ occurs in B and $t \sim_B s$ we need to show that also a sequent $\Gamma', p(s) \Longrightarrow \Delta'$ ocurs in B. We observe first that there is no rule that removes or changes an atomic formula occuring in a sequent. Even in eqLeft and eqRight the substituted formula is added. Therefore we will have a sequent $\Gamma'', t \doteq s, p(t) \Longrightarrow \Delta''$ in B. An application of eqLeft now completes the argument.

This completes the definition of the structure $\mathscr{M} = (H, \delta, I)$.

$$I(t) = [t]_b \text{ for every ground term } t. \qquad (1.7)$$ termInM

For 0-place function symbols c claim (1.7) is just the defintion of $I(c)$. The rest of the claim follows by an easy induction on the structural complexity of t.

The next phase in the proof of Lemma 1.34 consists in the verification of the claim

$$\mathscr{M} \models \bigwedge \Gamma \wedge \bigwedge_{F \in \Delta} \neg F \text{ for all sequents } s = \Gamma \Longrightarrow \Delta \text{ in } B \qquad (1.8)$$ ModelpropertyOfM

The proof of claim (1.8) is reduced to the following

For every formula ϕ
if there is $\Gamma \Longrightarrow \Delta \in B$ with $\phi \in \Gamma$ then $\mathscr{M} \models \phi$ (1.9) ModelpropertySingleFml
if there is $\Gamma \Longrightarrow \Delta \in B$ with $\phi \in \Delta$ then $\mathscr{M} \not\models \phi$

Claim (1.9) is proved by induction on the structural complexity $n(\phi)$ of ϕ If $n(\phi) = 0$ then there ϕ is an atomic formula or an equation.

For an atomic formula $p(\bar{t}) \in \Gamma$ we know by definition of $\mathscr{M}$ that $\mathscr{M} \models p(\bar{t})$. Now, consider $p(\bar{t}) \in \Delta$. If $\mathscr{M} \models p(\bar{t})$ then there must by definition of $\mathscr{M}$ be a sequent $\Gamma' \Longrightarrow \Delta'$ in B with $p(\bar{t}) \in \Gamma'$. Since atomic formulas never get removed we must have either $p(\bar{t}) \in \Delta$ and $p(\bar{t}) \in \Gamma$ or $p(\bar{t}) \in \Delta'$ and $p(\bar{t}) \in \Gamma'$. In both cases the branch B could be closed, contrary to assumption. Thus we must have $\mathscr{M} \models \neg p(\bar{t})$ for all $p(\bar{t}) \in \Delta$.

For $t_1 \doteq t_2 \in \Gamma$ we get $t_1 \sim_B t_2$ by definition of $\sim_B$. Thus $[t_1]_B = [t_2]_B$ which directly yields $\mathscr{M} \models t_1 \doteq t_2$.

For $t_1 \doteq t_2 \in \Delta$

The inductive step $n(\phi) > 0$ is split into a number of cases.

Case 1 $\neg\phi$ in Γ.
Since branch B is assumed to be *exhausted* rule notLeft will have been applied. There is thus a sequent $\Gamma' \Longrightarrow \Delta'$ in B with $\phi\Delta'$. By induction hypothesis we know $\mathscr{M} \not\models \phi$ thus $\mathscr{M} \models \neg\phi$.

Case 2 $\neg\phi$ in Δ.
Since branch B is assumed to be *exhausted* rule notRight will have been applied. There is thus a sequent $\Gamma' \Longrightarrow \Delta'$ in B with $\phi\Gamma'$. By induction hypothesis we know $\mathscr{M} \models \phi$ thus $\mathscr{M} \not\models \neg\phi$.

Case 3 $\phi_1 \wedge \phi_2$ in Γ.
Since branch B is assumed to be *exhausted* rule andLeft will have been applied. There is thus a sequent $\Gamma' \Longrightarrow \Delta'$ in B with $\phi_1, \phi_2\Gamma'$. By induction hypothesis we know $\mathscr{M} \models \phi_1$ and $\mathscr{M} \models \phi_2$ thus $\mathscr{M} \not\models \phi_1 \wedge \phi_2$.

Case 4 $\phi_1 \wedge \phi_2$ in Δ.
Since branch B is assumed to be *exhausted* rule andRight will have been applied. There is thus a sequent $\Gamma' \Longrightarrow \Delta'$ in B with $\phi_1\Delta'$ or $\phi_2\Delta'$. By induction hypothesis we know $\mathscr{M} \not\models \phi_1$ or $\mathscr{M} \not\models \phi_2$ thus in any case $\mathscr{M} \not\models \phi_1 \wedge \phi_2$.

Case 5 $\phi_1 \vee \phi_2$ in Γ.
Since branch B is assumed to be *exhausted* rule orLeft will have been applied. There is thus a sequent $\Gamma' \Longrightarrow \Delta'$ in B with $\phi_1\Gamma'$ or $\phi_2\Gamma'$. By induction hypothesis we know $\mathscr{M} \models \phi_1$ or $\mathscr{M} \models \phi_2$ thus in total $\mathscr{M} \models \phi_1 \vee \phi_2$.

Case 6 $\phi_1 \vee \phi_2$ in Δ.
Since branch B is assumed to be *exhausted* rule orRight will have been applied. There is thus a sequent $\Gamma' \Longrightarrow \Delta'$ in B with $\phi_1, \phi_2\Delta'$. By induction hypothesis we know $\mathscr{M} \not\models \phi_1$ and $\mathscr{M} \not\models \phi_2$ thus in total $\mathscr{M} \not\models \phi_1 \vee \phi_2$.

Case 7 $\phi_1 \to \phi_2$ in Γ.
Since branch B is assumed to be *exhausted* rule impLeft will have been applied. There is thus a sequent $\Gamma' \Longrightarrow \Delta'$ in B with $\phi_1 \in \Delta'$ or $\phi_2 \in \Gamma'$. By induction hypothesis we know $\mathscr{M} \not\models \phi_1$ or $\mathscr{M} \models \phi_2$ thus in any case $\mathscr{M} \models \phi_1 \to \phi_2$.

Case 8 $\phi_1 \to \phi_2$ in Δ.
Since branch B is assumed to be *exhausted* rule impRight will have been applied. There is thus a sequent $\Gamma' \Longrightarrow \Delta'$ in B with $\phi_1 \in \Gamma'$ and $\phi_2 \in \Delta'$. By induction hypothesis we know $\mathscr{M} \models \phi_1$ and $\mathscr{M} \not\models \phi_2$ thus in any case $\mathscr{M} \not\models \phi_1 \to \phi_2$.

Case 9 $\exists x.\phi$ in Γ.
Since branch B is assumed to be *exhausted* rule exLeft will have been applied. There is thus a sequent $\Gamma' \Longrightarrow \Delta'$ in B with $[x/c]\phi \in \Gamma'$. By induction hypothesis we know $\mathscr{M} \models [x/c]\phi$ and thus also $\mathscr{M} \models \exists x.\phi$.

Case 10 $\exists x.\phi$ in Δ.
Since branch B is assumed to be *exhausted* rule exRihgt will have been applied for every ground term t. For any ground term t there is thus a sequent $\Gamma' \Longrightarrow \Delta'$ in B with $[x/t]\phi \in \Delta'$. By induction hypothesis we know $\mathscr{M} \not\models [x/t]\phi$ for all such t. Since all elements in the universe of $\mathscr{M}$ are of the form $[t]_B$ for a ground term t we get $\mathscr{M} \not\models \exists x.\phi$.

Case 11 $\forall x.\phi$ in Γ.
Since branch B is assumed to be *exhausted* rule allLeft will have been applied for every ground term t. For any ground term t there is thus a sequent $\Gamma' \Longrightarrow \Delta'$ in B with $[x/t]\phi \in \Gamma'$. By induction hypothesis we know $\mathscr{M} \models [x/t]\phi$ for all such t. Since all elements in the universe of $\mathscr{M}$ are of the form $[t]_B$ for a ground term t we get $\mathscr{M} \models \forall x.\phi$.

Case 12 $\forall x.\phi$ in Δ.
Since branch B is assumed to be *exhausted* rule allRight will have been applied. There is thus a sequent $\Gamma' \Longrightarrow \Delta'$ in B with $[x/c]\phi \in \Delta'$. By induction hypothesis we know $\mathscr{M} \not\models [x/c]\phi$ and thus also $\mathscr{M} \not\models \forall x.\phi$.

This completes the proof of claim (1.9) and thus also of claim (1.8).

Since any branch necessarily contains the root node we have $S \in B$ and by claim (1.8) $\mathscr{M} \not\models S$ which contradicts the assumed univeral validity of sequent S. Thus the assumption that there is no closed proof tree with root labeled by S has been refuted. This, finally, completes the proof of Lemma 1.34 and thus the proof of Theorem 1.21.

Here are some closing remarks. Lemma 1.31 states that all rules of the calculus are sound and complete. Soundness of rules was explicitely used in the soundness proof of the calculus, proof of Lemma 1.33. On the other hand, in the proof of the completeness Lemma 1.34, no reference to Lemma 1.31 or the completeness of rules was made. So, what is the significance of rules to be complete? The rule

$$\text{orLeftIncompl}\ \frac{\Gamma, \phi \Longrightarrow \Delta}{\Gamma, \phi \vee \psi \Longrightarrow \Delta}$$

is sound but not complete. If we apply it during proof search, from bottom to top, we might be lucky and be able to close the proof. Or, we might fail and would have to backtrack and apply the other half of the orLeft rule. This is the significance of complete rules, they always lead in the right direction and need never to be reconsidered.

Finally we come back to the discussion, already aluded to after Figure 1.2, of the differences between the calculus presented in these notes and the calculus implemented in the KeY system. Since the later contains less rules for a more restricted language soundness follows directly from Lemma 1.33. There is no easy argument that the completeness of the implemented calculus follows from Lemma 1.34. The best way is to look again in the details of the proof of Lemma 1.34 and observe that the rule eqDynamicSort is only used when an equation $t_1 \doteq t_2$ occurs with the types of t_1, t_2 being incompatibly. In the implemented calculus thus is not allowed and rule eqDynamicSort is thus not needed.

There is another difference between the two calculi that best be discussed here, although a syntax extension is involved that will onlylater be introduced in Figure 1.3. The provisions $\sigma(t_1) \sqsubseteq \sigma(t_2)$ in eqLeft and $\sigma(t_2)$ and $\sigma(t_1)$ in eqRight are relaxed to $[z/t_2](\phi)$, repsectively $[z/t_1](\phi)$ is well-typed. The problem with the example presented earlier persists. We recall the example with two types $A \neq B$ with $B \sqsubseteq A$, two constant symbols $a : \to A$ and $b : \to B$, and a unary predicate $p(B)$. Applying eqLeft with the relaxed restriction on the sequent $b \doteq a, p(b) \Longrightarrow$ would result in $b \doteq a, p(b), p(a) \Longrightarrow$ with $p(a)$ ill-typed. The calculus presented in these notes would simply not apply rule eqLeft in this case. The calculus implemented in the KeY system would automatically introduce a cast and result in $b \doteq a, p(b), p(cast_B(a)) \Longrightarrow$. Unfortunately, this is not explicit in the rules but part of the rule applications formalism. Since with this trick more rules are available then without it completeness is not an issue. On the other hand it can be easily seen that this implicit rule extension is sound. We will demonstrate the argument with the above example. We start with $cast_B(b) \doteq cast_B(b)$ by eqReflLeft. From $b \doteq a$ we get by eqLeft $cast_B(b) \doteq cast_B(a)$ and by axiom (Ax-C) (see Figure 1.8) $b \doteq cast_B(a)$. Now, rule eqLeft applied to $b \doteq a, p(b)$ yields $p(cast_B(a))$. As a derived rule from a set of sound rules the implicit rule is also sound. We trust that the reader knows how to contruct the general argument from the given example.

End of Digression on Completeness Proof

1.2.6 Digression On Ill-typed Formulas

This is another loose end totally unrelated to other parts of this technical report. But, I find it an interesting observation.

The rule **eqLeft** in Figure 1.2 is only sound with the side condition $[z/t_2](\phi)$ is well-typed. More precisely, if this side condition is dropped formulas can be derived that are not welltyped, i.e. are not formulas at all. In this subsection we try to investigate how serious this is. Can this lead to an inconsistent logic?

Let Σ be a signature for a given type hierarchy $\mathcal{T}$. By $\Sigma_\perp$ we denote the signature for the type hierarchy $\{\perp\}$ that differs from Σ only in the typing of the function and relation symbols. Since there is only one type the typing in $\Sigma_\perp$ is obvious. It amounts to an untyped signature. Note, as far as the syntax is concerned there is no difference between $(\mathcal{T}, \Sigma)$ formulas and $(\{\perp\}, \Sigma_\perp)$ formulas.

By $\vdash_\Sigma$ we denote the derivability relation for $(\mathcal{T}, \Sigma)$ formulas. Likewise $\vdash_\perp$ stands for the derivability relation for $(\{\perp\}, \Sigma_\perp)$ formulas. By $\vdash^0_\Sigma$ we denote the derivability relation for $(\mathcal{T}, \Sigma)$ formulas when all typing side conditions in the rules (taclets) are ignored. It is $\vdash^0_\Sigma$ that we want to investigate. For this it suffices to observe that $\vdash^0_\Sigma$ coincides with $\vdash_\perp$. Since we know that $\vdash_\perp$ is sound, $\vdash^0_\Sigma$ is also sound.

Thus we conclude, that dropping the typing side conditions in the proof rules allows to derive formulas that most likely will not help towards closing the proof, but soundness is not compromised.

End Of Digression On Ill-typed Formulas

1.3 Extended First-Order Logic

sec02:ExtFOL

In this section we extend the Basic First-Order Logic from Section 1.2. First we turn our attention in Subsection 1.3.1 to an additional term building construct: *variable binders*. They do not increase the expressive power of the logic, but are extremely handy.

An issue that comes up in almost any practical use of logic, are partial functions. In the KeY system, partial functions are treated via underspecification as explained in Subsection 1.3.2. In essence this amounts to replacing a partial function by all its extensions to total functions.

1.3.1 Variable Binders

subsec02:VarBinders

This subsection assumes that the type *int* of mathematical integers, the type *LocSet* of sets of locations, and the type *Seq* of finite sequences are present in TSym. For the logic JFOL to be presented in Subsection 1.4 this will be obligatory.

A typical example of a variable binder symbol is the sum operator, as in $\Sigma_{k=1}^{n} k^2$. Variable binders are related to quantifiers in that they *bind* a variable. The KeY system does not provide a generic mechanism to include new binder symbols. Instead we list the binder symbols included at the moment.

A more general account of binder symbols is contained in the doctoral thesis Ulbrich13 [Ulbrich, 2013, Subsection 2.3.1]. Binder symbols do not increase the expressive power of first-order logic: for any formula ϕ_b containing binder symbols there is a formula ϕ without such that ϕ_b is universally valid if and only if ϕ is, see Ulbrich13 [Ulbrich, 2013, Theorem 2.4]. This is the reason why one does not find binder symbols other than quantifiers in traditional first-order logic text books.

Definition 1.35 (extends Definition 1.3).

4. If *vi* is a variable of type *int*, b_0, b_1 are terms of type *int* not containing *vi* and *s* is an arbitrary term in Trm_{int}, then $bsum\{vi\}(b_0,b_1,s)$ is in Trm_{int}.
5. If *vi* is a variable of type *int*, b_0, b_1 are terms of type *int* not containing *vi* and *s* is an arbitrary term in Trm_{int}, then $bprod\{vi\}(b_0,b_1,s)$ is in Trm_{int}.
6. item:infUnionDefbinder If *vi* is a variable of arbitrary type and *s* a term of type *LocSet*, then $\mathit{infiniteUnion}\{vi\}(s)$ is in Trm_{LocSet}.

item:seqDefbinder

def:BinderSymbols

7. If vi is a variable of type *int*, b_0, b_1 are terms of type *int* not containing vi and s is an arbitrary term in Trm_{any}, then $\quad seqDef\{vi\}(b_0, b_1, s) \quad$ is in Trm_{Seq}.

It is instructive to observe the role of the quantified variable vi in the following syntax definition:

def:FreeBoundVars2

Definition 1.36 (extends Definition 1.5). If t is one of the terms $bsum\{vi\}(b_0, b_1, s)$, $bprod\{vi\}(b_0, b_1, s)$, $infiniteUnion\{vi\}(s)$, and $seqDef\{vi\}(b_0, b_1, s)$ we have

$$var(t) = var(b_0) \cup var(b_1) \cup var(s) \text{ and } fv(t) = var(t) \setminus \{vi\} \ .$$

We trust that the following remarks will suffice to clarify the semantic meaning of the first two symbols introduced in Definition 1.35. In mathematical notation one would write $\Sigma_{b_0 \le vi < b_1} s_{vi}$ for $bsum\{vi\}(b_0, b_1, s)$ and $\Pi_{b_0 \le vi < b_1} s_{vi}$ for $bprod\{vi\}(b_0, b_1, s)$. For the corner case $b_1 \le b_0$ we stipulate $\Sigma_{b_0 \le vi < b_1} s_{vi} = 0$ and $\Pi_{b_0 \le vi < b_1} s_{vi} = 1$. The name *bsum* stands for *bounded sum* to emphasize that infinite sums are not covered. The proof rules for *bsum* and *bprod* are the obvious recursive definitions plus the stipulation for the corner cases which we forgo to reproduce here.

For an integer variable vi the term $infiniteUnion\{vi\}(s)$ would read in mathematical notation $\bigcup_{-\infty < vi < \infty} s$, and analogously for variables vi of type other than integer. The precise semantics is part of Figure 1.12 in Section 1.4.4 below.

The semantics of $seqDef\{vi\}(b_0, b_1, s)$ will be given in Definition 1.53 on page 61. But, it makes an interesting additional example of a binder symbol. The term $seqDef\{vi\}(b_0, b_1, s)$ is to stand for the finite sequence $\langle s(b_0), s(b_0 + 1), \ldots, s(b_1 - 1)\rangle$. For $b_1 \le b_0$ the result is the empty sequence, i.e., $seqDef\{vi\}(b_0, b_1, s) = \langle\rangle$. The proof rules related to *seqDef* are discussed in Chapter 5 Section 2 in the KeY book.

1.3.2 Undefinedness

subsec02:Undefinedness

In KeY all functions are total. There are two ways to interpret a function symbol f in a structure $\mathscr{M}$ at an argument position $\bar{a}$ outside its intended range of definition:

1. The value of the function $val_{\mathscr{M}}(f)$ at position $\bar{a}$ is set to a default within the intended range of f. E.g., $bsum\{vi\}(1, 0, s)$ evaluates to 0 (regardless of s).
2. The value of the function $val_{\mathscr{M}}(f)$ at position $\bar{a}$ is set to an arbitrary value b within the intended range of f. For different structures different b are chosen. When we talk about universal validity, i.e., truth in all structures, we assume that for every possible choice of b there is a structure $\mathscr{M}_b$ such that $val_{\mathscr{M}_b}(f)(\bar{a}) = b$. The prime example for this method, called *underspecification*, is division by 0 such that, e.g., $\frac{1}{0}$ is an arbitrary integer.

Another frequently used way to deal with undefinedness is to choose an error element that is different from all defined values of the function. We do not do this.

The advantage of underspecification is that no changes to the logic are required. But, one has to know what is happening. In the setting of underspecification we can prove $\exists\, i; (\frac{1}{0} \doteq i)$ for an integer variable i. However, we cannot prove $\frac{1}{0} \doteq \frac{2}{0}$. Also the formula $cast_{int}(c) \doteq 5 \rightarrow c \doteq 5$ is not universally valid. In case c is not of type *int* the underspecified value for $cast_{int}(c)$ could be 5 for $c \neq 5$.

The underspecification method gives no warning when undefined values are used in the verification process. The KeY system offers a well-definedness check for JML contracts, details are described in Section 8.3.3 in the KeY book.

1.4 First-Order Logic for Java

sec02:JavaFOL

As already indicated in the introduction of this chapter, Java first-order logic (JFOL) will be an instantiation of the extended classical first-order logic from Subsection 1.3 tailored towards the verification of Java programs. The precise type hierarchy $\mathscr{T}$ and signature Σ will of course depend on the program and the statements to be proved about it. But we can identify a basic vocabulary that will be useful to have in almost every case. Figure 1.3 shows the type hierarchy $\mathscr{T}_J$ that we require to be at least contained in the type hierarchy $\mathscr{T}$ of any instance of JFOL. The mandatory function and predicate symbols Σ_J are shown in Figure 1.4. Data types are essential for formalizing nontrival program properties. The data types of the integers and the theory of arrays are considered so elementary that they are already included here. More precisely what is covered here are the mathematical integers. There are of course also Java integers types. Those and their relation to the mathematical integers are covered in Section on Integers in the KeY book. Also the special data type *LocSet* of sets of memory locations will already be covered here. Why it is essential for the verification of Java programs will become apparent in Chapters 8 and 9 in the KeY book . The data type of *Seq* of finite sequences however will extensively be treated later in Section 1.5.

1.4.1 Type Hierarchy and Signature

subsec02:JFOLSignature

The mandatory type hierarchy $\mathscr{T}_J$ for JFOL is shown in Figure 1.3. Between *Object* and *Null* the class and interface types from the Java code to be investigated will appear. In the future there might be additional data types at the level immediately below *Any* besides *boolean*, *int*, *LocSet* and *Seq*, e.g., *maps*.

The mandatory vocabulary Σ_J of JFOL is shown in Figure 1.4 using the same notation as in Definition 1.2. In the subsections to follow we will first present the axioms that govern these data types one by one and conclude with their model-theoretic semantics in Subsection 1.4.6.

As mentioned above, in the verification of a specific Java program the signature Σ may be a strict superset of Σ_J. To mention just one example: for every model field m

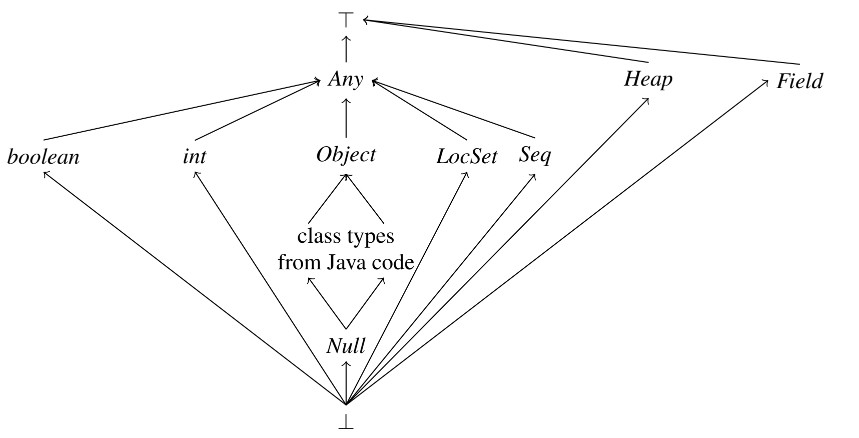

Fig. 1.3 The mandatory type hierarchy $\mathscr{T}_J$ of JFOL fig:typeHierarchy

of type T contained in the specification of a Java class C an new symbol $f_m : Heap \times C \to T$ is introduced. The KeY book uses the terminology that function symbols with at least one, usually the first, argument of type *Heap* are called *observer function symbols*.

int and *boolean*	all function and predicate symbols for *int*, e.g., $+, *, <, \ldots$
	boolean constants *TRUE*, *FALSE*
Java types	$null : Null$
	$length : Object \to int$
	$cast_A : Object \to A$ for any A in $\mathscr{T}$ with $\bot \sqsubset A \sqsubseteq Object$.
	$instance_A : Any \to boolean$ for any type $A \sqsubseteq Any$
	$exactInstance_A : Any \to boolean$ for any type $A \sqsubseteq Any$
Field	$created : Field$
	$arr : int \to Field$
	$f : Field$ for every Java field f
Heap	$select_A : Heap \times Object \times Field \to A$ for any type $A \sqsubseteq Any$
	$store : Heap \times Object \times Field \times Any \to Heap$
	$create : Heap \times Object \to Heap$
	$anon : Heap \times LocSet \times Heap \to Heap$
	$wellFormed(Heap)$
LocSet	$\varepsilon(Object, Field, LocSet)$
	$empty, allLocs : LocSet$
	$singleton : Object \times Field \to LocSet$
	$subset(LocSet, LocSet)$
	$disjoint(LocSet, LocSet)$
	$union, intersect, setMinus : LocSet \times LocSet \to LocSet$
	$allFields : Object \to LocSet,\ allObjects : Field \to LocSet$
	$arrayRange : Object \times int \times int \to LocSet$
	$unusedLocs : Heap \to LocSet$

Fig. 1.4 The mandatory vocabulary Σ_J of JFOL fig:SigmaJ

1.4.2 Axioms for Integers

subsec02:JFOLAxiomsInt

polySimp_addComm0	$k+i \doteq i+k$	add_zero_right	$i+0 \doteq i$
polySimp_addAssoc	$(i+j)+k \doteq i+(j+k)$	add_sub_elim_right	$i+(-i) \doteq 0$
polySimp_elimOne	$i*1 \doteq i$	mul_distribute_4	$i*(j+k) \doteq (i*j)+(i*k)$
mul_assoc	$(i*j)*k \doteq i*(j*k)$	mul_comm	$j*i \doteq i*j$
less_trans	$i<j \wedge j<k \rightarrow i<k$	less_is_total_heu	$i<j \vee i \doteq j \vee j<i$
less_is_alternative_1	$\neg(i<j \wedge j<i)$	less_literals	$0<1$
add_less	$i<j \rightarrow i+k<j+k$	multiply_inEq	$i<j \wedge 0<k \rightarrow i*k<j*k$

$$\text{int\_induction}\ \frac{\Gamma \Longrightarrow \phi(0), \Delta \quad \Gamma \Longrightarrow \forall n; (0 \leq n \wedge \phi(n) \rightarrow \phi(n+1)), \Delta}{\Gamma \Longrightarrow \forall n; (0 \leq n \rightarrow \phi(n)), \Delta}$$

Fig. 1.5 Integer axioms and rules fig:intrules

Figure 1.5 shows the axioms for the integers with $+$, $*$ and $<$. Occasionally we use the additional symbol $\leq$ which is, as usual, defined by $x \leq y \leftrightarrow (x < y \vee x \doteq y)$. The implication multiply_inEq does in truth not occur among the KeY taclets. Instead multiply_inEq0 $i \leq j \wedge 0 \leq k \rightarrow i*k \leq j*k$ is included. But, multiply_inEq can be derived from , multiply_inEq0 although by a rather lengthy proof (65 steps) based on a normal form transformation. The reverse implication is trivially true.

Figure 1.5 also lists in front of each axiom the name of the taclet that implements it. The KeY system not only implements the shown axioms but many useful consequences and defining axioms for further operations such as those related to integer division and the modulo function. How the various integer data types of the Java language are handled in the KeY system is explained in Section 5.4 of the KeY book.

sidebar:Incompleteness

Incompleteness

Mathematically the integers $(\mathbb{Z}, +, *, 0, 1, <)$ are a commutative ordered ring satisfying the well-foundedness property: every nonempty subset of the positive integers has a least element. Well-foundedness is a second-order property. It is approximated by the first-order induction schema, which can be interpreted to say that every nonempty definable subset of the positive integers has a least element. The examples known so far of properties of the integers that can be proved in second-order logic but not in its first-order approximation, see e.g. [Kirby and Paris, 1982] KirbyParis82 are still so arcane that we need not worry about this imperfection.

1.4.3 Axioms for Heap

subsec02:JFOLAxiomsHeap

The state of a Java program is determined by the values of the local variables and the heap. A heap assigns to every pair consisting of an object and a field declared for this object an appropriate value. As a first step to model heaps, we require that a type *Field* be present in JFOL. This type is required to contain the field constant *created* and the fields $arr(i)$ for array access for natural numbers $0 \leq i$. In a specific verification context there will be constants f for every field f occurring in the Java program under verification. There is no assumption, however, that these are the only elements in *Field*; on the contrary, it is completely open which other field elements may occur. This feature is helpful for modular verification: when the contracts for methods in a Java class are verified, they remain true when new fields are added. The data type *Heap* allows us to represent more functions than can possibly occur as heaps in states reachable by a Java program:

1. Values may be stored for arbitrary pairs (o, f) of objects o and fields f regardless of the question if f is declared in the class of o.
2. item:typmismatch The value stored for a pair (o, f) need not match the type of f.
3. A heap may assign values for infinitely many objects and fields.

On one hand our heap model allows for heaps that we will never need, on the other hand this generality makes the model simpler. Relaxation 2 in the above list is necessary since JFOL does not use dependent types. To compensate for this shortcoming there has to be a family of observer functions $select_A$, where A ranges over all subtypes of *Any*.

The axiomatization of the data type *Heap*, shown in Figure 1.6, follows the pattern well known from the theory of arrays. The standard reference is McCarthy62 [McCarthy, 1962]. There are some changes however. One would expect the following rule $select_A(store(h,o,f,x),o2,f2) \leadsto$ if $o \doteq o2 \wedge f \doteq f2$ then x else $select_A(h,o2,f2)$. Since the type of x need not be A this easily leads to an ill-typed formula. Thus we need $cast_A(x)$ in place of x. In addition the implicit field *created* gets special treatment. The value of this field should not be manipulated by the *store* function. This explains the additional conjunct $f \dot{\neq} created$ in the axiom. The rule selectOfStore as it is shown below implies $select_A(store(h,o,created,x),o2,f2) \doteq select_A(h,o2,f2)$. Assuming extensionality of heaps this entails $store(h,o,created,x) \doteq h$. The *created* field of a heap can only be changed by the *create* function as detailed by the rule selectOfCreate. This ensures that the value of the *created* field can never be changed from *TRUE* to *FALSE*. Note also, that the object *null* is considered to be created from the start, so it can be excepted from rule selectOfCreate.

There is another operator, named $anon(h,s,h')$, that returns a *Heap* object. Its meaning is described by the rule selectOfAnon in Figure 1.6: at locations (o, f) in the location set s the resulting heap coincides with h' under the proviso $f \dot{\neq} created$, otherwise it coincides with h. To get an idea when this operator is useful imaging that h is the heap reached at the beginning of a while loop that at most changes locations in a location set s and that h' is a totally unknown heap. Then $anon(h,s,h')$ represents a heap reached after an unknown number of loop iterations. This heap may

selectOfStore	$select_A(store(h,o,f,x),o2,f2) \rightsquigarrow$ if $o \doteq o2 \wedge f \doteq f2 \wedge f \dot{\neq} created$ then $cast_A(x)$ else $select_A(h,o2,f2)$
selectOfCreate	$select_A(create(h,o),o2,f) \rightsquigarrow$ if $o \doteq o2 \wedge o \dot{\neq} null \wedge f \doteq created$ then $cast_A(TRUE)$ else $select_A(h,o2,f)$
selectOfAnon	$select_A(anon(h,s,h'),o,f) \rightsquigarrow$ if$(\varepsilon(o,f,s) \wedge f \dot{\neq} created) \vee \varepsilon(o,f,unusedLocs(h)))$ then $select_A(h',o,f)$ else $select_A(h,o,f)$

with the typing $o,o1,o2 : Object, f, f2 : Field, h,h' : Heap, s : LocSet$

Fig. 1.6 Rules for the theory of arrays. fig:ArrayTheoryrules

have more created objects than the initial heap h. Since location sets are not allowed to contain locations with not created objects, see onlyCreatedObjectsAreInLocSets in Figure 1.7, this has to be added as an addition case in rule selectOfAnon. This application scenario also accounts for the name which is short for *anonymize*.

A patiently explained example for the use of *store* and *select* functions can be found in Subsection 15.2.3 of the KeY book . While SMT solvers can handle expressions containing many occurrences of *store* and *select* quite efficiently, they are a pain in the neck for the human reader. The KeY interface therefore presents those expressions in a pretty printed version, see explanations in Sectio 16.2 of the KeY book.

The taclets in Figure 1.6 are called *rewriting taclets*. We use the $\rightsquigarrow$ notation to distinguish them from the other sequent rules as, e.g., in Figures 1.1 and 1.2. A rewriting rule $s \rightsquigarrow t$ is shorthand for a sequent rule $\frac{\Gamma' \Longrightarrow \Delta'}{\Gamma \Longrightarrow \Delta}$ where $\Gamma' \Longrightarrow \Delta'$ arises from $\Gamma \Longrightarrow \Delta$ by replacing one or more occurrences of the term s by t. Rewriting rules will again be discussed in the taclet tutorial of the KeY book.

Our concept of heap is an overgeneralization. Most of the time this does no harm. But, there are situations where it is useful to establish and depend on certain well-formedness conditions. The predicate $wellFormed(heap)$ has been included in the vocabulary for this purpose. No effort is made to make the $wellFormed(h)$ predicate so strong that it only is true of heaps h that can actually occur in Java programs. The axioms in Figure 1.7 were chosen on a pragmatic basis. There is e.g., no axiom that guarantees for a created object o of type A with $select(h,o,f)$ defined that the field f is declared in class A.

The first four axioms in Figure 1.7 formalize properties of well-formed heaps while the rest cover situations starting out with a well-formed heap, manipulate it and end up again with a well-formed heap. The formulas are quite self-explanatory. Reading though them you will encounter the auxiliary predicate `arrayStoreValid`: `arrayStoreValid(o,x)` is true if o is an array object of exact type $A[]$ and x is of type A.

The meaning of the functions symbols $instance_A(x)$, $exactInstance_A(x)$, $cast_A(x)$, and $length(x)$ is given by the axioms in Figure 1.8. This time we present the axioms in mathematical notation for conciseness. The axiom scheme, (Ax-I) and (Ax-C)

onlyCreatedObjectsAreReferenced
$wellFormed(h) \to select_A(h,o,f) \doteq null \lor select_{boolean}(h, select_A(h,o,f), created) \doteq TRUE$

onlyCreatedObjectsAreInLocSets
$wellFormed(h) \land \varepsilon(o2, f2, select_{LocSet}(h,o,f)) \to o2 \doteq null \lor$
$select_{boolen}(h, o2, created) \doteq TRUE$

narrowSelectType
$wellFormed(h) \land select_B(h,o,f) \to select_A(h,o,f)$ where type of f is A and $A \sqsubseteq B$

narrowSelectArrayType
$wellFormed(h) \land o \not\doteq null \land select_B(h,o,arr(i)) \to select_A(h,o,arr(i))$
where type of o is $A[]$ and $A \sqsubseteq B$

wellFormedStoreObject
$wellFormed(h) \land (x \doteq null \lor (select_{boolean}(h,x,created) \doteq TRUE \land instance_A(x) \doteq TRUE))$
$\to wellFormed(store(h,o,f,x))$ where type of f is A

wellFormedStoreArray
$wellFormed(h) \land (x \doteq null \lor (select_{boolean}(h,x,created) \doteq TRUE \land arrayStoreValid(o,x)))$
$\to wellFormed(store(h,o,arr(idx),x)))$

wellFormedStoreLocSet
$wellFormed(h) \land \forall ov; \forall fv; (\varepsilon(ov,fv,y) \to ov \doteq null \lor select_{boolean}(h,ov,created) \doteq TRUE)$
$\to wellFormed(store(h,o,f,y))$ where type of f is A and $LocSet \sqsubseteq A$

wellFormedStorePrimitive
$wellFormed(h) \to wellFormed(store(h,o,f,x))$
provided f is a field of type A, x is of type B, and $B \sqsubseteq A, B \not\sqsubseteq Object, B \not\sqsubseteq LocSet$

wellFormedStorePrimitiveArray
$wellFormed(h) \to wellFormed(store(h,o,arr(idx),x))$
provided o is of sort A, x is of sort $B, B \not\sqsubseteq Object, B \not\sqsubseteq LocSet, B \sqsubseteq A$

wellFormedCreate
$wellFormed(h) \to wellFormed(create(h,o))$

wellFormedAnon
$wellFormed(h) \land wellFormed(h2) \to wellFormed(anon(h,y,h2))$

In the above formulas the following implicitly universally quantified variables are used: $h, h2 : Heap$, $o, x : Object$, $f : Field$, $i : int$, $y : LocSet$

Fig. 1.7 Rules for the predicate *wellFormed* fig:wellFormedRules

show that adding $instance_A$ and $cast_A$ does not increase the expressive power. These functions can be defined already in the basic logic plus underspecification. The formulas (Ax-E_1) and (Ax-E_2) completely axiomatize the $exactInstance_A$ functions, see Lemma 1.45 on page 52. The function $length$ is only required to be not negative. Axioms (Ax-E_1), (Ax-E_2), and (Ax-L) are directly formalized in the KeY system as taclets instance_known_dynamic_type, exact_instance_known_dynamic_type and arrayLengthNotNegative. The other two axioms families have no direct taclet counterpart. But, they can easily be derived.

$$\forall Object\ x; (instance_A(x) \doteq TRUE \leftrightarrow \exists y; (y \doteq x)) \text{ with } y : A \qquad \text{(Ax-I)}$$

eq:ax-i

$$\forall Object\ x; (exactInstance_A(x) \doteq TRUE \rightarrow instance_A(x) \doteq TRUE) \qquad \text{(Ax-E}_1\text{)}$$

eq:ax-e1

$$\forall Object\ x; (exactInstance_A(x) \doteq TRUE \rightarrow instance_B(x) \doteq FALSE) \text{ with } A \not\sqsubseteq B \qquad \text{(Ax-E}_2\text{)}$$

eq:ax-e2

$$\forall Object\ x; (instance_A(x) \doteq TRUE \rightarrow cast_A(x) \doteq x) \qquad \text{(Ax-C)}$$

eq:ax-c

$$\forall Object\ x; (length(x) \geq 0) \qquad \text{(Ax-L)}$$

Fig. 1.8 Axioms for functions related to Java types

fig:AxiomsTypRelated

1.4.4 Axioms for Location Sets

subsec02:JFOLAxiomsLocSet

The data type *LocSet* is a very special case of the set type in that only sets of heap locations are considered, i.e., sets of pairs (o, f) with o an object and f a field. This immediately guarantees that the is-element-of relation ε is well-foundedfor *LocSet*. Problematic formulas such as $a\varepsilon a$ are already syntactically impossible.

The rules for the data type *LocSet* are displayed in Figure 1.9. The only constraint on the membership relation ε is formulated in rule equalityToElementOf. One could view this rule as a definition of equality for location sets. But, since equality is a built in relation in the basic logic it is in fact a constraint on ε. All other rules in this figure are definitions of the additional symbols of the data type, such as, e.g., *allLocs*, *union*, *intersect*, and $infiniteUnion\{av\}(s1)$.

elementOfEmpty	$\varepsilon(o1, f1, empty)$	$\rightsquigarrow FALSE$
elementOfAllLocs	$\varepsilon(o1, f1, allLocs)$	$\rightsquigarrow TRUE$
equalityToElementOf	$s1 \doteq s2$	$\rightsquigarrow \forall o; \forall f; (\varepsilon(o, f, s1) \leftrightarrow \varepsilon(o, f, s2))$
elementOfSingleton	$\varepsilon(o1, f1, singleton(o2, f2)$	$\rightsquigarrow o1 \doteq o2 \wedge f1 \doteq f2$
elementOfUnion	$\varepsilon(o1, f1, union(t1, t2))$	$\rightsquigarrow \varepsilon(o1, f1, t1) \vee \varepsilon(o1, f1, t2)$
subsetToElementOf	$subset(t1, t2)$	$\rightsquigarrow \forall o; \forall f; (\varepsilon(o, f, t1) \rightarrow \varepsilon(o, f, t2))$
elementOfIntersect	$\varepsilon(o1, f1, intersect(t1, t2))$	$\rightsquigarrow \varepsilon(o1, f1, t1) \wedge \varepsilon(o1, f1, t2)$
elementOfAllFields	$\varepsilon(o1, f1, allFields(o2)$	$\rightsquigarrow o1 \doteq o2$
elementOfSetMinus	$\varepsilon(o1, f1, setMinus(t1, t2))$	$\rightsquigarrow \varepsilon(o1, f1, t1) \wedge \neg\varepsilon(o1, f1, t2)$
elementOfAllObjects	$\varepsilon(o1, f1, allObjects(f2)$	$\rightsquigarrow f1 \doteq f2$
elementOfInfiniteUnion	$\varepsilon(o1, f1, infiniteUnion\{av\}(s1))$	$\rightsquigarrow \exists av; \varepsilon(o1, f1, s1)$

with the typing $o, o1, 02 : Object, f, f1 : Field, s1, s2, t1, t2 : LocSet$, av of arbitrary type.

Fig. 1.9 Rules for data type *LocSet*

fig:LocSetrules

1.4.5 Digression On The Theory of Integers

We return to a more thorough account of the axioms on integers and types presented so far.

Mathematically the integers are a nontrivial ordered commutative ring with unit that satisfies the least number principle. While references to Peano's axioms for the natural numbers abound, it is hard to find at least one reference to an axiom system for integers. The axioms shown in Figure 1.10 appear in unpublished lecture notes Greenleaf00 Greenleaf [2000-2008].

Comparison with KeY Taclets

We will show that the theory given by the axioms in Figure 1.10 is the same as the theory given by those in Figure 1.5. More precisely, we will show that all axioms in Figure 1.5 are derivable from the axioms in Figure 1.10 and vice versa, all axioms in Figure 1.10 are derivable from the axioms in Figure 1.5. We first consider the obvious implications.

$A.1$	$(i+j)+k \doteq i+(j+k)$
$A.2$	$i+j \doteq j+i$
$A.3$	$0+i \doteq i$
$A.4$	$i+(-i) \doteq 0$
$M.1$	$(i*j)*k \doteq i*(j*k)$
$M.2$	$i*j \doteq j*i$
$M.3$	$1*x \doteq x$
$M.4$	$i*(j+k) \doteq i*j+i*k$
$M.5$	$1 \neq 0$
$O.1$	$0 < i \vee 0 = i \vee 0 < (-i)$
$O.2$	$0 < i \wedge 0 < j \to 0 < i+j$
$O.3$	$0 < i \wedge 0 < j \to 0 < i*j$
$O.4$	$i < j \leftrightarrow 0 < j+(-i)$
$O.5$	$\neg(0 < 0)$
Ind	$\phi(0) \wedge \forall i(0 \leq i \wedge \phi(i) \to \phi(i+1)) \to \forall i(0 \leq i \to \phi(i))$

Fig. 1.10 Integer Axioms fig:intaxioms

lem:intax2intrules1 **Lemma 1.37.** *The following rules from Figure 1.5 are derivable from the axioms in Figure 1.10:*

add_zero_right	*polySimp_addComm0*
polySimp_addAssoc	*add_sub_elim_right*
polySimp_elimOne	*mul_distribute_4*
mul_assoc	*mul_comm*
less_trans	*less_is_alternative_1*
int_induction	

Proof.

add_zero_right	follows from $A.3$ and $A.2$.
polySimp_addComm0	equals $A.2$
polySimp_addAssoc	equals $A.1$
add_sub_elim_right	equals $A.4$
polySimp_elimOne	follows from $M.3$ and $M.2$.
mul_distribute_4	equals $M.4$
mul_assoc	equals $M.1$
mul_comm	equals $M.2$

less_trans

$$\begin{array}{rll} i < j \wedge j < k & \Leftrightarrow 0 < j + (-i) \wedge 0 < k + (-j) & O.4 \\ & \Rightarrow 0 < (j + (-i)) + (k + (-j)) & O.2 \\ & \Leftrightarrow 0 < k + (-i) + (j + (-j)) & A.1, A.2 \\ & \Rightarrow 0 < k + (-i) & A.3, A.4 \\ & \Leftrightarrow i < k & O.4 \end{array}$$

less_is_alternative_1

$$\begin{array}{rll} i < j \wedge j < i & \Leftrightarrow 0 < j + (-i) \wedge 0 < i + (-j) & O.4 \\ & \Rightarrow 0 < (j + (-i)) + (i + (-j)) & O.2 \\ & \Leftrightarrow 0 < 0 & A.1 - A.4 \end{array}$$

By $O.5$ this yields $\neg(i < j \wedge j < i)$.

int_induction follows directly from *Ind*

That was easy. The derivations of the remaining six rules from Figure 1.5 require much more effort. We start with the a first set of consequences from these axioms.

lem:IntAxConsequences1

Lemma 1.38. *The following formulas can be derived from the axioms in Figure 1.10:*

item:minusUnique *C.1.* $i + j \doteq 0 \to j \doteq -i$
item:minusminus *C.2.* $-(-i)) \doteq i$
item:zeroUnique *C.3.* $\forall j(i + j \doteq j) \to i \doteq 0$
item:minuszero *C.4.* $-0 \doteq 0$
item:minussum *C.5.* $-(i + j) \doteq (-i) + (-j)$
item:timeszero *C.6.* $0 * i \doteq 0$
item:multminusone *C.7.* $-i \doteq (-1) * i$
item:oneUnique *C.8.* $\forall j(i * j \doteq j) \to i \doteq 1$
item:minusonesquared *C.9.* $(-1) * (-1) \doteq 1$
item:minusdistr *C.10.* $(-i) * j \doteq -(i * j)$

Proof. **ad** C.1

$$\begin{array}{rll} i + j & \doteq 0 & \\ (i + j) + (-i) & \doteq 0 + (-i) & \text{equational logic} \\ (i + (-i)) + j & \doteq 0 + (-i) & A.1 \text{ and } A.2 \\ (i + (-i)) + j & \doteq -i & A.3 \\ j & \doteq -i & A4 \text{ and } A.3 \end{array}$$

ad C.2

$$\begin{array}{rll} i + (-i) & \doteq 0 & A4 \\ (-i) + i & \doteq 0 & A2 \\ i & \doteq -(-i) & C.1 \end{array}$$

ad C.3

$$\begin{array}{ll} \forall j(i + j \doteq j) & \text{assumption} \\ i + 0 \doteq 0 & \text{instantiation for } \forall j \\ i \doteq 0 & A2 \text{ and } A.3 \end{array}$$

$$\begin{array}{lllll}
\textbf{ad C.4} & 0+(-0) & \doteq 0 & A.4 \\
 & 0+(-0) & \doteq -0 & A.3 \\
 & -0 & \doteq 0 & \text{equational logic}
\end{array}$$

$$\begin{array}{llll}
\textbf{ad C.5} & (i+j)+(-i+(-j)) & \doteq (i+(-i))+(j+(-j)) & A.1,A.2 \\
 & & \doteq 0+0 & A.4 \\
 & & \doteq 0 & A.3
\end{array}$$

Now, C.1 yields $-(i+j) \doteq -i+(-j)$.

ad C.6 We first observe:

$$\begin{array}{lll}
0*i & \doteq (0+0)*i & A.3 \\
 & \doteq 0*i+0*i & M.4,A.2
\end{array}$$

We start a new line of reasoning:

$$\begin{array}{lll}
0 & \doteq 0*i+-(0*i) & A.4 \\
 & \doteq (0*i+0*i)+-(0*i) & \text{above equation} \\
 & \doteq 0*i+(0*i+-(0*i) & A.1 \\
 & \doteq 0*i+0 & A.4 \\
 & \doteq 0*i & A.2,A.2
\end{array}$$

$$\begin{array}{llll}
\textbf{ad C.7} & i+(-1)*i & \doteq 1*i+(-1)*i & M.3 \\
 & & \doteq (1+(-1))*i & M.4,A.2 \\
 & & \doteq 0*i & A.4 \\
 & & \doteq 0 & C.6
\end{array}$$

Now, C.1 yields $-i \doteq (-1)*i$.

$$\begin{array}{lll}
\textbf{ad C.8} & \forall j(i*j \doteq j) & \text{assumption} \\
 & i*1 \doteq 1 & \text{instantiation for } \forall j \\
 & i \doteq 1 & M.3,A.2
\end{array}$$

$$\begin{array}{llll}
\textbf{ad C.9} & -(-1) & \doteq (-1)*(-1) & C.7 \\
 & 1 & \doteq (-1)*(-1) & C.2
\end{array}$$

$$\begin{array}{llll}
\textbf{ad C.10} & i*j+(-i)*j & \doteq (i+(-i))*j & M.4,A.2 \\
 & & \doteq 0*j & A.4 \\
 & & \doteq 0 & C.6
\end{array}$$

By C.1 we get $-(i*j) \doteq (-i)*j$.

lem:intax2intrules2

Lemma 1.39. *From the axioms in Figure 1.10 the rules are derivable from the axioms in Figure 1.5*

$$\begin{array}{ll}
\mathit{less\_is\_total\_heu} & i<j \lor i=j \lor j<i \\
\mathit{less\_literals} & 0<1 \\
\mathit{add\_less} & i<j \rightarrow i+k<j+k \\
\mathit{multiply\_inEq} & i<j \land 0<k \rightarrow i*k<j*k
\end{array}$$

Proof. less_is_total_heu

$$\begin{array}{ll}
0<j+(-i) \lor 0=j+(-i) \lor 0<-(j+(-i)) & O.1 \\
0<j+(-i) \lor i=(j+(-i))+i \lor 0<-(j+(-i)) & \text{equational logic} \\
0<j+(-i) \lor i=j \lor 0<-(j+(-i)) & A.1,A.4,A.2,A.3 \\
0<j+(-i) \lor i=j \lor 0<(-j)+-(-i)) & C.5 \\
0<j+(-i) \lor i=j \lor 0<i+(-j) & A.2,C.2 \\
i<j \lor i=j \lor j<i & O.4
\end{array}$$

less_literals By $O.1$ we know $0 < 1 \vee 0 = 1 \vee 0 < (-1)$. By $M.5$ this reduces to $0 < 1 \vee 0 < (-1)$. If $0 < 1$ we are finished. So assume $0 < (-1)$. From $M.3$ we get $0 < (-1) * (-1)$. Now, C.9 yields $0 < 1$.

add_less

$$\begin{array}{lll} i+k < j+k & \Leftrightarrow 0 < (j+k) + -(i+k) & O.4 \\ & \Leftrightarrow 0 < (j+k) + ((-i) + (-k)) & C.5 \\ & \Leftrightarrow 0 < (j+(-i)) + (k+(-k)) & A.1, A.2 \\ & \Leftrightarrow 0 < j+(-i) & A.3, A.4 \\ & \Leftrightarrow i < j & O.4 \end{array}$$

multiply_inEq

$$\begin{array}{lll} i < j \wedge 0 < k & \Leftrightarrow 0 < j + (-i) \wedge 0 < k & O.4 \\ & \Rightarrow 0 < (j+(-i)) * k & O.3 \\ & \Leftrightarrow 0 < j*k + (-i)*k & M.4, A.2 \\ & \Leftrightarrow 0 < j*k + (-(i*k)) & C.10 \\ & \Leftrightarrow i*k < j*k & O.4 \end{array}$$

Having finished the proofs of Lemmas 1.37 and 1.39 we now turn to the reverse implication.

lem:intrules2intax **Lemma 1.40.** *From the rules in Figure 1.5 all axioms in Figure 1.10 can be derived.*

Proof.

$A.1$ $(i+j)+k \doteq i+(j+k)$	equals p*olySimp_addAssoc*
$A.2$ $i+j \doteq j+i$	equals p*olySimp_addComm0*
$A.3$ $0+i \doteq i$	follows from a*dd_zero_right* and $A.2$
$A.4$ $i+(-i) \doteq 0$	equals a*dd_sub_elim_right*
$M.1$ $(i*j)*k \doteq i*(j*k)$	equals m*ul_assoc*
$M.2$ $i*j \doteq j*i$	equals m*ul_comm*
$M.3$ $1*x \doteq x$	follows from p*olySimp_elimOne* and $M.2$
$M.4$ $i*(j+k) \doteq i*j+i*k$	equals m*ul_distribute_4*
$M.5$ $1 \neq 0$	follows from l*ess_literals* and l*ess_base*
$O.1$ $0 < i \vee 0 = i \vee 0 < (-i)$	follows from l*ess_is_total_heu*
$O.2$ $0 < i \wedge 0 < j \to 0 < i+j$	$0 < j \Rightarrow i < i+j$ by a*dd_less* and $A.2$
	$0 < i \wedge i < i+j \Rightarrow 0 < i+j$ by l*ess_trans*
$O.3$ $0 < j \wedge 0 < k \to 0 < j*k$	follows from m*ultiply_inEq*
$O.4$ $i < j \leftrightarrow 0 < j+(-i)$	follows from a*dd_less* plus $A.1, A.4$
	for $\to$ instantiate $k = -i$
	for $\leftarrow$ instantiate $k = i$
$O.5$ $\neg(0 < 0)$	follows from l*ess_base*
Ind $\phi(0) \wedge \forall i(0 \leq i \wedge \phi(i) \to \phi(i+1)) \to \forall i(0 \leq i \to \phi(i))$	equals i*nt_induction*

Derived Properties

There are many useful properties of the integers that can derived from the given axioms. Let us start with some simple examples.

lem:intSimpleDerived **Lemma 1.41.**

item:lessIrrefl *1.* $\neg(i < i)$
item:immSuccPlusOne *2.* $i < i + 1$
item:lessInvMultMinus *3.* $i < j \to -j < -i$

Proof.
(**a**d1) can easily be seen to follow from axiom less_is_alternative_1 $\neg(i < j \wedge j < i)$ by taking $j = i$. By the way, the formula $\neg(i < i)$ is contained among the KeY taclets by the name less_base.
(**a**d2) can be seen to follow from less_literals $0 < 1$, add_less, and add_zero_right.
(**a**d3) From $i < j$ we obtain by applying add_less twice $(i + (-i)) + (-j) < (j + (-i)) + (-j)$, which by associativity, commutativity, axioms add_sub_elim_right and add_zero_right yields the desired conclusion.

Upto now we have tried to stick to the vocabulary of the axoms. For ease of use it is indispensible to introduce definitional extensions, e.g.

$$i \leq j \leftrightarrow (i = j \vee i < j) \tag{1.10}$$ DefLEQ

A consequence that is not immediately obvious is the claim in the following lemma.

lem:intImmSucc **Lemma 1.42.** *The formula*

$$\forall i. \forall j. (i < j \to i + 1 \leq j)$$

can be derived from the axioms in Figure 1.5.
In words we could say that $i + 1$ is the immediate successor of i.

Proof. We first show by induction j

$$0 < j \to 1 \leq j \tag{1.11}$$ immSuccPart1

The induction formula $\phi(j)$ in the notation of Figure 1.5 is $0 < j \to 1 \leq j)$. The base case $0 < 0 \to 1 \leq 0$ is true since by less_base the formula $\neg(0 < 0)$ can be derived. To show $0 \leq j \wedge \phi(j) \to \phi(j+1)$ we assume $0 \leq j$, $0 < j \to 1 \leq j$ and $0 < j + 1$ with the aim to show $1 \leq j + 1$. From axiom less_is_total_heu we obtain $0 = j \vee 0 < j \vee j < 0$. In case $0 = j$ the claim to be proved, tacitly using add_zero_right, reduces to $1 \leq 1$ which is true by definition. In case $0 < j$ we get $1 \leq j$ from the induction hypothesis and $1 \leq j + 1$ by (2) and transitivity. The last alternative $j < 0$ and the assumption $0 \leq j$ would yield by transitivity $j < j$ contradicting (1). This yields $\forall j. (0 \leq j \to (0 < j \to 1 \leq j))$. Simple propositional reasoning and the definition of $\leq$ now prove (1.11).
The claim of the lemma is now obtained by the following chain of reasoning:

$i < j$	lefthand side
$i+(-i) < j+(-i)$	add_less
$0 < j+(-i)$	add_sub_elim_right
$1 \leq j+(-i)$	(1.11)
$1+i \leq (j+(-i))+i$	add_less and Def.of $\leq$
$i+1 \leq j+(i+(-i))$	associativity and commutativity of $+$
$i+1 \leq j$	add_sub_elim_right

Justification of the Axioms

It is a well-know result of mathemcatical logic that there can be no complete axiomatization for the natural numbers. This result can easily be extended to show that there can be no complete axiomatization for the intgers. This raises the question of how close the axioms shown in Figure 1.10 describe the integers. In this paragraph we try to answer this question.

By Int^2 we denote the axiom system from 1.10 with the decisive change that the induction axiom Ind is replaced by the second order induction axiom Ind^2 where X ranges over arbitrary subsets of the respective domain:

$$\forall X(0 \in X \wedge \forall x((0 \leq x \wedge x \in X) \rightarrow (x+1) \in X) \rightarrow \forall x(0 \leq x \rightarrow x \in X))$$

thm:Int2UniqueModel

Theorem 1.43. *Any two models of Int^2 are isomorphic.*

Proof

Let $\mathscr{A}$, $\mathscr{B}$ be two models of Int^2. By s^A, s^B we denote the interpretation of symbol s in $\mathscr{A}$ and $\mathscr{B}$. Thus 0^A, 1^A are the additive and multiplicative unit in $\mathscr{A}$, while 0^B, 1^B are the corresponding units in $\mathscr{B}$. For $n \geq 1$ we denote by t_n for purposes of this proof the term $1+\ldots+1$, that consists of n-fold addition of of the constant 1. Furthermore, t_0 is simply the constant 0. For brevity we use n^A, n^B to denote the interpretation of the term t_n in $\mathscr{A}$ and $\mathscr{B}$.

Let $C_A = \{n^A \mid n \geq 0\}$. An appeal to Ind^2 shows

$$\{a \in A \mid \mathscr{A} \models 0 \leq a\} = C_A$$

Sideremark: We have written sloppily $\mathscr{A} \models 0 \leq a$ for the correct but cumbersome statement $(\mathscr{A}, \beta) \models 0 \leq x$ with $\beta(x) = a$.

Likewise we have $C_B = \{n^B \mid n \geq 0\} = \{b \in B \mid \mathscr{B} \models 0 \leq b\}$.

This will be the only two applications of second order induction Ind^2, in all other applications the first-order scheme Ind will suffice.

In the following contruction of the isomorphism π we will list the formulas that are need at each step. After finishing the explanations we turn to showing that all these formulas can be derived from the axioms.

Using

$$\forall i(i < 0 \to 0 < -i) \tag{1.12}$$ align:neg2pos

and

$$\forall i(-(-i) = i) \tag{1.13}$$ align:minusInvolution

we see that the universe A of structure $\mathscr{A}$ equals $\{n^A \mid n \geq 0\} \cup \{-^A(n^A) \mid n < 0\}$ and likewise $B = \{n^B \mid n \geq 0\} \cup \{-^B(n^B) \mid n < 0\}$.

It will later prove handy to extend the notation introduced above a bit and use t_{-n} to stand for the term $-t_n$ for $n \geq 1$ and $(-n)^A$ and $(-n)^B$ to stand for the interpretations $(-t_n)^{\mathscr{A}}$ and $(-t_n)^{\mathscr{B}}$. In this notation the description of the universes given above look like this $A = \{n^A \mid n \in \mathbb{Z}\}$ and $B = \{n^B \mid n \in \mathbb{Z}\}$.

We define the mapping $\pi : A \to B$ by $\pi(n^A) = n^B$ and $\pi(-^A n^A) = -^B n^B$. For this to be a sound definition we need

$$\neg(t_n \doteq t_m) \quad \text{for all } 0 \leq m,n \text{ with } n \neq m \tag{1.14}$$ align:posdiff

and

$$\neg\exists x(0 \leq x \wedge x < 0) \tag{1.15}$$ align:posDisjointNeg

Notice that (1.13) and (1.14) together imply $\neg(-t_n \doteq -t_m)$ for all $0 \leq m,n$ with $n \neq m$.

The fact that the consequences (1.12) to (1.15) from the axioms are true in $\mathscr{A}$ guarantees that π is well-defined. Given the representation of B the definition of π ensures that it is surjective. The fact that the consequences (1.13) to (1.14) from the axioms are true in $\mathscr{B}$ guarantees that π is injective. It remains to prove the homomorphism properties

item:hom0 1. $\pi(0^A) = 0^B$
item:hom1 2. $\pi(1^A) = 1^B$
item:homplus 3. $\pi(a_1 +^A a_2) = \pi(a_1) +^B \pi(a_2)$ for $a_1, a_2 \in A$
item:homtimes 4. $\pi(a_1 *^A a_2) = \pi(a_1) *^B \pi(a_2)$ for $a_1, a_2 \in A$
item:homminus 5. $\pi(-^A a) = -^B \pi(a)$ for $a \in A$
item:homorder 6. $a_1 <^A a_2) \Leftrightarrow \pi(a_1) <^B \pi(a_2)$ for $a_1, a_2 \in A$

The requirements 1 and 2 follow directly from the definition.

The remaining requirements could be infered from the following formulas respectively:

$$t_n + t_m \doteq t_{n+m} \tag{1.16}$$ align:homplus

$$t_n * t_m \doteq t_{n*m} \tag{1.17}$$ align:homtimes

$$-t_m \doteq t_{-m} \tag{1.18}$$ align:homminus

$$t_n < t_m \text{ for } n,m \in \mathbb{Z} \text{ with } n < m \tag{1.19}$$ align:homorder

and

$$\neg(t_n < t_m)\text{for } n,m \in \mathbb{Z} \text{ with } n \not< m \tag{1.20}$$ align:homorderNeg

Proof of 1.12:

Instantiating j with 0 we obtain from $O.4$ and $A.3$ the equivalence $i < 0 \leftrightarrow 0 < -i$, as required.

Proof of 1.13:

We first prove

$$\text{For every } i \text{ there is a unique element } j \text{ with } j + i = 0. \tag{1.21}$$ align:minusunique

Assume there are j, j' such that $j + i = j' + i = 0$. Then

$$j = 0 + j = (j' + i) + j = j' + (i + j) = j' + 0 = j'$$

as required.

$$\forall i(-(-i) \doteq i) \tag{1.22}$$ align:minusminus

From axiom $A.3$ $i + (-i) = 0$ statement (1.21) leads to $-(-i) = i$. We tacitely use commutativity all the time.

Proof of 1.14:

From $O.3$ and $O.4$ we obtain $\forall i(\neg(i < i))$ or logical equivalently $\forall i \forall j(i < j \to i \neq j)$. From Lemma 1.39 we know $0 < 1$ which gives us $\neg(0 \doteq 1)$ for a start. From $0 < 1$ we may derive $0 < t_n$ for every $n > 0$ by repeated application of $O.2$. Thus we know $\neg(0 \doteq t_n)$ for every $n > 0$. For $0 \leq n < m$ we derive $t_m + -t_n \doteq t_{m-n}$ using associativity, commutativity and $-(i + j) \doteq -i + -j$. This last equality follows from $(-i + -j) + i + j = 0$ and the uniqueness of the additive inverse, observed above. In the situation $0 \leq n < m$ we have $0 < m - n$ and thus as seen above can derive $0 < t_{m-n}$, which leads to $0 < t_m + -t_n$. Now, $O.4$ yields derivability of $t_n < t_m$ and thus $\neg(t_n \doteq t_m)$.

We note for the verification of 1.19 below that we have already shown that $t_n < t_m$ is derivable for $0 \leq n < m$.

Proof of 1.15:

By Lemma 1.37 we know $\forall i(0 \leq i \wedge i < 0 \rightarrow 0 < 0)$, which by axiom $O.5$ yields $\forall i(\neg(0 \leq i \wedge i < 0)$. This, of course, is equivalent to $\neg\exists i(0 \leq i \wedge i < 0)$.

Proof of 1.16:

Case $\mathbf{0} \leq \mathbf{m}, \mathbf{n}$ In this case $t_n + t_m$ and t_{n+m} are modulo associativity the same term.
Case $\mathbf{m}, \mathbf{n} < \mathbf{0}$ Using the previous case and the equation $-(i+j) \doteq -i + -j$ already established above we get the following line of reasoning:

$$t_n + t_m \doteq -t_{-n} + -t_{-m} \doteq -(t_{-n} + t_{-m}) \doteq -(t_{-(n+m)}) \doteq t_{n+m}$$

Case $\mathbf{m} < \mathbf{0}, \mathbf{n} \geq \mathbf{0}$
Making use of the equation $-(i+j) \doteq -i + -j$ noted above the term $t_n + t_m$ is of the form $1 + \ldots 1 + (-1) + \ldots + (-1)$. Using associativity, commutativity and $A.3$, $A.4$ this can equivalently be reduced to
Subcase $-\mathbf{m} \leq \mathbf{n}$: t_{n+m} of the form $1 + \ldots + 1$.
Subcase $-\mathbf{m} > \mathbf{n}$ $(-1) + \ldots + (-1)$, which using $-(i+j) \doteq -i + -j$ again is equivalent to t_{n+m}.
So, in both cases we have established the claim.
Case $\mathbf{n} < \mathbf{0}, \mathbf{m} \geq \mathbf{0}$ Dual to the previous case.

Proof of 1.17:

Before we delve into the details let us look at a tpyical example: we need to show that e.g., $(1+1) * (1+1+1) \doteq 1+1+1+1+1+1$ is derivable.

We need some preliminaries to do this.

$$\forall i(0 \doteq 0 * i) \tag{1.23}$$

align:timeszero

Here is a proof for 1.23:

$$\begin{aligned}
0 &\doteq 0 * i + -(0 * i) && A.4 \\
&\doteq (0+0) * i + -(0 * i) && A.3 \\
&\doteq (0 * i + 0 * i) + -(0 * i) && M.2, M.4 \\
&\doteq 0 * i + (0 * i + -(0 * i) && A.1 \\
&\doteq 0 * i + 0 && A.4 \\
&\doteq 0 * i && A.3
\end{aligned}$$

$$\forall i \forall j(j * (-i) \doteq -(j * i)) \tag{1.24}$$

align:timesminus

Proof for 1.24: First look at:

$$\begin{aligned} j*i+j*(-i) &= j*(i+(-i)) && M.4 \\ &= j*0 && A.4 \\ &= 0 && (1.23) \end{aligned}$$

Since the additive inverse is unique (see above) we must have $-(j*i) \doteq j*(-i)$.
Case $\mathbf{0 \leq m, n}$ This we prove by induction.
initial case:

$$\begin{aligned} t_n * t_0 &\doteq t_n * 0 && \text{Def. of } t_0 \\ &\doteq 0 && 1.23 \\ &\doteq t_{n*0} && \text{Def. of } t_0 \end{aligned}$$

step case

$$\begin{aligned} t_n * t_{m+1} &\doteq t_n * (t_m + 1) && \text{Def. of } t_{m+1} \\ &\doteq t_n * t_m + t_n * 1 && M.4 \\ &\doteq t_n * t_m + t_n && M.3 \\ &\doteq t_{n*m} + t_n && \text{Ind. Hyp.} \\ &\doteq t_{n*m+n} && \text{Def. of } t_{n*m+n} \\ &\doteq t_{n*(m+1)} && \text{metalevel reasoning} \end{aligned}$$

Case $\mathbf{m, n < 0}$

$$\begin{aligned} t_n * t_m &\doteq (-t_{-n}) * (-t_{-m}) && \text{Def. of } t_n \\ &\doteq -(-(t_{-n} * t_{-m})) && (1.24) \text{ twice} \\ &\doteq t_{-n} * t_{-m} && (1.22) \\ &\doteq t_{(-n)*(-m)} && \text{previous case} \\ &\doteq t_{n*m} && \text{metalevel reasing} \end{aligned}$$

Case $\mathbf{m \geq 0, n < 0}$

$$\begin{aligned} t_n * t_m &\doteq (-t_{-n}) * t_m && \text{Def. of } t_n \\ &\doteq -(t_{-n} * t_m) && (1.24) \\ &\doteq -(t_{(-n)*m}) && \text{first case above} \\ &\doteq t_{-(-n)*m} && (1.24) \\ &\doteq t_{n*m} && \text{metalevel reasing} \end{aligned}$$

Case $\mathbf{m < 0, n \geq 0}$ analogous to previous case.

Proof of 1.18:

Before we go on we note the following derivable property

$$\forall i \forall j((-i) < (-j) \leftrightarrow j < i) \tag{1.25}$$

align:minusorder

$$\begin{aligned} (-i) < (-j) &\leftrightarrow 0 < (-j) + -(-(i) && O.4 \\ &\leftrightarrow 0 < i + (-j) && (1.22) \\ &\leftrightarrow j < i && O.4 \end{aligned}$$

Case $\mathbf{0} \leq \mathbf{n}, \mathbf{m}$ This case we have already covered above in the proof of (1.14).
Case $\mathbf{n}, \mathbf{m} < \mathbf{0}$ Can be reduced by (1.25) to the first case.
Case $\mathbf{m} < \mathbf{0}, \mathbf{0} \leq \mathbf{n}$ From the first case we get the derivability of $0 \leq t_n$ and $0 < t_{-m}$. Now, (1.25 we obtain further $-t_{-m} < 0$ which by definition of t_m for negative m is the same as $t_m < 0$. By the transitivity of $<$ proved in Lemma 1.37 we get $t_m < t_n$ as desired.
Case $\mathbf{n} < \mathbf{0}, \mathbf{0} \leq \mathbf{m}$ Symmetric to the previous case.

Proof of 1.20:

We again start with a simple observation:

$$\forall i(\neg(i < i)) \tag{1.26}$$

align:notIdorder

By $O.4$ the relation $i < i$ is equivalent to $0 < i + (-i)$ and thus to $0 < 0$, which contradicts $O.5$.

Now, if $\neg(n < m)$ then by Lemma 1.39 either $n = m$ in which case we can derive $\neg(k_n < k_n)$, by (1.26) or $m < n$. In the second case we can derive $t_m < t_n$ as seen in (1.18). If we also had $t_n < t_m$ then by transitivity the contradiction $t_n < t_n$ would follow. Thus we can derive $\neg(t_n < t_m)$. □

cor:Int2complete

Corollary 1.44. *For any formula ϕ in the vocabulary of the theory Int:*

$$Int^2 \vdash \phi \quad \Leftrightarrow \quad \mathbb{Z} \models \phi$$

Proof

Since $\mathbb{Z}$ is a model of the theory Int^2 the implication $\Rightarrow$ is true. For the reverse implication assume $\mathbb{Z} \models \phi$ and $Int^2 \not\vdash \phi$. Then there is a model $\mathscr{M}$ of Int^2 with $\mathscr{M} \models \neg\phi$. By Theorem 1.43 **M** is isormorphic to $\mathbb{Z}$.Contradiction! □

Corollary 1.44 states that Int^2 axiomatizes the theory of the integers completely. We take this as an indication that the first-order relaxation *Int* of Int^2 is a good approximation of the theory of the integers. Note, that the second-order induction axioms is only used once in the proof of Theorem 1.43.

End: Digression On The Theory of Integers

1.4.6 Semantics

subsec02:JFOLSemantics

As already remarked at the start of Subsection 1.2.3, a formal semantics opens up the possibility for rigorous soundness and relative completeness proofs. Here we extend and adapt the semantics provided there to cover the additional syntax introduced for JFOL (see Section 1.4.1).

We take the liberty to use an alternative notion for the interpretation of terms. While we used $\mathrm{val}_{\mathscr{M},\beta}(t)$ in Section 1.2.3 to emphasize also visually that we are concerned with evaluation, we will write $t^{\mathscr{M},\beta}$ for brevity here.

The definition of a FOL structure $\mathscr{M}$ for a given signature in Subsection 1.2.3 was deliberately formulated as general as possible, to underline the universal nature of logic. The focus in this subsection is on semantic structures tailored towards the verification of Java programs. To emphasize this perspective we call these structures JFOL structures.

A decisive difference to the semantics from Section 1.2.3 is that now the interpretation of some symbols, types, functions, predicates, is constrained. Some functions are completely fixed, e.g., addition and multiplication of integers. Others are almost fixed, e.g., integer division n/m that is fixed except for $n/0$ which may have different interpretations in different structures. Other symbols are only loosely constrained, e.g., *length* is only required to be nonnegative.

The semantic constraints on the JFOL type symbols are shown in Figure 1.11. The restriction on the semantics of the subtypes of *Object* is that their domains contain for every $n \in \mathbb{N}$ infinitely many elements o with $length^{\mathscr{M}}(o) = n$. The reason for this is the way object creation is modeled. When an array object is created an element o in the corresponding type domain is provided whose *created* field has value *FALSE*. The created field is then set to *TRUE*. Since the function *length* is independent of the heap it cannot be changed in the creation process. So, the element picked must already have the desired length. This topic will be covered in detail in Subsection 3.6.6 of the KeY book. The semantics of *Seq* will be given in a later chapter .

- $D^{int} = \mathbb{Z}$,
- $D^{boolean} = \{tt, ff\}$,
- $D^{ObjectType}$ is an infinite set of elements for every *ObjectType* with $Null \sqsubset ObjectType \sqsubseteq Object$,
  subject to the restriction that for every positive integer n there are infinitely many elements o in $D^{ObjectType}$ with $length^{\mathscr{M}}(o) = n$.
- $D^{Null} = \{null\}$,
- D^{Heap} = the set of all functions $h : D^{Object} \times D^{Field} \to D^{Any}$,
- D^{LocSet} = the set of all subsets of $\{(o, f) \mid o \in D^{Object}$ and $f \in D^{Field}\}$,
- D^{Field} is an infinite set.

Fig. 1.11 Semantics on type domains fig:FixedTypedomains

Constant Domain

Let T be a theory, that does not have finite models. By definition $T \vdash \phi$ iff $\mathscr{M} \models \phi$ for all models $\mathscr{M}$ of T. The Löwenheim-Skolem Theorem, which by the way follows easily from the usual completeness proofs, guarantees that $T \vdash \phi$ iff $\mathscr{M} \models \phi$ for all countably infinite models $\mathscr{M}$ of T. Let S be an arbitrary countably infinite set, then we have further $T \vdash \phi$ iff $\mathscr{M} \models \phi$ for all models $\mathscr{M}$ of T such that the universe of $\mathscr{M}$ is S. To see this assume there is a countably infinite model $\mathscr{N}$ of T with universe N such that $\mathscr{N} \models \neg\phi$. For cardinality reasons there is a bijection b from N onto S. So far, S is just a set. It is straightforward to define a structure $\mathscr{M}$ with universe S such that b is an isomorphism from $\mathscr{N}$ onto $\mathscr{M}$. This entails the contradiction $\mathscr{M} \models \neg\phi$.

The interpretation of all the JFOL function and predicate symbols listed in Figure 1.4 is at least partly fixed. All JFOL structures $\mathscr{M} = (M, \delta, I)$ are required to satisfy the constraints put forth in Figure 1.12.

Some of these constraints are worth an explanation. The semantics of the *store* function, as stated above, is such that it cannot change the implicit field *created*. Also there is no requirement that the type of the value x should match with the type of the field f. This liberality necessitates the use of the $cast_A$ functions in the semantics of $select_A$.

It is worth pointing out that the *length* function is defined for all elements in D^{Object}, not only for elements in D^{OT} where OT is an array type.

Since the semantics of the wellFormed predicate is a bit more involved we put it separately in Figure 1.13

The integer operations are defined as usual with the following versions of integer division and the modulo function:

$$n /^{\mathscr{M}} m = \begin{cases} \text{the uniquely defined } k \text{ such that} & \\ |m| * |k| \leq |n| \text{ and } |m| * (|k| + 1) > |n| \text{ and} & \\ k \geq 0 \text{ if } m, n \text{ are both positive or both negative and} & \\ k \leq 0 \text{ otherwise} & \text{if } m \neq 0 \\ & \\ \text{unspecified} & \text{otherwise} \end{cases}$$

Thus integer division is a total function with arbitrary values for $x /^{\mathscr{M}} 0$. Division is an example of a partially fixed function. The interpretation of $/$ in a JFOL structure $\mathscr{M}$ is fixed except for the values $x /^{\mathscr{M}} 0$. These may be different in different JFOL structures. The modulo function is defined by

$$\text{mod}(n, d) = n - (n/d) * d$$

Note, that this implies $\text{mod}(n, 0) = n$ as $/$ is – due to using underspecification – a total function.

1. $TRUE^{\mathcal{M}} = tt$ and $FALSE^{\mathcal{M}} = ff$
2. $select_A^{\mathcal{M}}(h,o,f) = cast_A^{\mathcal{M}}(h(o,f))$
3. $store^{\mathcal{M}}(h,o,f,x) = h^*$, where the function h^* is defined by
$$h^*(o',f') = \begin{cases} x & \text{if } o' = o, f = f' \text{ and } f \neq created^{\mathcal{M}} \\ h(o',f') & \text{otherwise} \end{cases}$$
4. $create^{\mathcal{M}}(h,o) = h^*$, where the function h^* is defined by
$$h^*(o',f) = \begin{cases} tt & \text{if } o' = o, o \neq null \text{ and } f = created^{\mathcal{M}} \\ h(o',f) & \text{otherwise} \end{cases}$$
5. $arr^{\mathcal{M}}$ is an injective function from $\mathbb{Z}$ into $Field^{\mathcal{M}}$
6. $created^{\mathcal{M}}$ and $f^{\mathcal{M}}$ for each Java field f are elements of $Field^{\mathcal{M}}$, which are pairwise different and also not in the range of $arr^{\mathcal{M}}$.
7. $null^{\mathcal{M}} = null$
8. $cast_A^{\mathcal{M}}(o) = \begin{cases} o & \text{if } o \in A^{\mathcal{M}} \\ \text{arbitrary element in } A^{\mathcal{M}} & \text{otherwise} \end{cases}$
9. $instance_A(o)^{\mathcal{M}} = tt \Leftrightarrow o \in A^{\mathcal{M}} \Leftrightarrow \delta(o) \sqsubseteq A$
10. $exactInstance_A^{\mathcal{M}} = tt \Leftrightarrow \delta(o) = A$
11. $length^{\mathcal{M}}(o) \in \mathbb{N}$
12. $\langle o,f,s \rangle \in \varepsilon^{\mathcal{M}}$ iff $(o,f) \in s$
13. $empty^{\mathcal{M}} = \emptyset$
14. $allLocs^{\mathcal{M}} = Object^{\mathcal{M}} \times Field^{\mathcal{M}}$
15. $singleton^{\mathcal{M}}(o,f) = \{(o,f)\}$
16. $\langle s_1,s_2 \rangle \in subset^{\mathcal{M}}$ iff $s_1 \subseteq s_2$
17. $\langle s_1,s_2 \rangle \in disjoint^{\mathcal{M}}$ iff $s_1 \cap s_2 = \emptyset$
18. $union^{\mathcal{M}}(s_1,s_2) = s_1 \cup s_2$
19. $infiniteUnion\{av\}(s)^{\mathcal{M}} = \{(a \in D^T \mid s^{\mathcal{M}}[a/av]\}$ with T type of av
20. $intersect^{\mathcal{M}}(s_1,s_2) = s_1 \cap s_2$
21. $setMinus^{\mathcal{M}}(s_1,s_2) = s_1 \setminus s_2$
22. $allFields^{\mathcal{M}}(o) = \{(o,f) \mid f \in Field^{\mathcal{M}}\}$
23. $allObjects^{\mathcal{M}}(f) = \{(o,f) \mid o \in Object^{\mathcal{M}}\}$
24. $arrayRange^{\mathcal{M}}(o,i,j) = \{(o, arr^{\mathcal{M}}(x) \mid x \in \mathbb{Z}, i \leq x \leq j\}$
25. $unusedLocs^{\mathcal{M}}(h) = \{(o,f) \mid o \in Object^{\mathcal{M}}, f \in Field^{\mathcal{M}}, o \neq null, h(o, created^{\mathcal{M}}) = false\}$
26. $anon^{\mathcal{M}}(h_1,s,h_2) = h^*$, where the function h^* is defined by:
$$h^*(o,f) = \begin{cases} h_2(o,f) & \text{if } (o,f) \in s \text{ and } f \neq created^{\mathcal{M}}, \text{ or} \\ & \quad (o,f) \in unusedLocs^{\mathcal{M}}(h_1) \\ h_1(o,f) & \text{otherwise} \end{cases}$$

Fig. 1.12 Semantics for the mandatory JFOL vocabulary (see Figure 1.4) fig:JFOLstructures

$$h \in wellFormed^{\mathcal{M}} \text{ iff } \begin{array}{ll} (a) & \text{if } h(o,f) \in D^{Object} \text{ then } h(o,f) = null \text{ or } h(h(o,f), created^{\mathcal{M}}) = tt \\ (b) & \text{if } h(o,f) \in D^{LocSet} \text{ then } nh(o,f) \cap unusedLocs^{\mathcal{M}}(h) = \emptyset \\ (c) & \text{if } \delta(o) = T[] \text{ then } \delta(h(o, arr^{\mathcal{M}}(i))) \sqsubseteq T \text{ for all } 0 \leq i < length^{\mathcal{M}}(o) \\ (d) & \text{there are only finitely many } o \in D^{Object} \text{ for which } h(o, created^{\mathcal{M}}) = tt \end{array}$$

Fig. 1.13 Semantics for the predicate *wellFormed* fig:JFOLstructuresWF

lem:TypeRelSoundComplete **Lemma 1.45.** *The axioms in Figure 1.8 are sound and complete with respect to the given semantics.*

For a proof see SchmittUlbrich15 [Schmitt and Ulbrich, 2015].

1.4.7 Digression on the Proof of Lemma 1.45

For the convenience of the reader we we include a proof of Lemma 1.45. This is an extended version of the proof published in SchmittUlbrich15 [Schmitt and Ulbrich, 2015].

We denote the axioms from Figure 1.8 by $ThJ_{\mathscr{T}}$.

We split the proof of Lemma 1.45 into a soundness and a completeness part.

lem:TypAxSound **Lemma 1.46.** *The axioms $ThJ_{\mathscr{T}}$ are sound with respect to the semantics from Figures 1.11 and 1.12.*

Proof. Let $\mathscr{T}^*$ be an extension of the type hierarchy $\mathscr{T} = (\text{TSym}, \sqsubseteq)$ and $\mathscr{M} = (M, \delta, I)$ be an arbitrary JFOL structure for $(\mathscr{T}*, \Sigma)$. We show $\mathscr{M} \models Ax$ for all axioms Ax in $ThJ_{\mathscr{T}}$.
ad 1.4.3 Fix an arbitary element $o \in M$.
Applying the semantics definition to the lefthand side yields $o \in instance_A^{\mathscr{M}} \Leftrightarrow \delta(o) \sqsubseteq A$. The righthand side is equivalent to the existence of a witness for the existential quantifier $\exists y$ i.e. to the existence of $o' \in A^{\mathscr{M}}$ with $o = o'$. This is itself equivalent to $o \in A^{\mathscr{M}}$, i.e., to $\delta(o) \sqsubseteq A$.
ad 1.4.3 Fix an arbitary element $o \in M$.

$$\begin{array}{llll} o \in exactInstance_A^{\mathscr{M}} & \Leftrightarrow \delta(o) = A & & \text{definitinon} \\ & \Rightarrow \delta(o) \sqsubseteq A & & \text{obvious} \\ & \Leftrightarrow o \in instance_A^{\mathscr{M}} & & \text{definitinon} \end{array}$$

ad1.4.3 Fix an arbitary element $o \in M$ and $B \in \text{TSym}$ with $A \not\sqsubseteq B$. We have to show taht we cannot have both $o \in exactInstance_A^{\mathscr{M}}$ and $o \in instance_B^{\mathscr{M}}$. Applying the definition in both cases yields $\delta(o) = A$ and $\delta(o) \sqsubseteq B$. Together we obtain $A \sqsubseteq B$, contradicting the choice of B.
ad 1.4.3 Is an immediate consequence of the definition of the $cast_A$ functions.

We now turn to the completeness part of the proof.

If a $(\mathscr{T}, \Sigma)$-structure $\mathscr{M}$ satisfies $\mathscr{M} \models ThJ_{\mathscr{T}}$, then it need not be a *Java* structure. The axioms would, e.g., be true if $exactInstance_A^{\mathscr{M}} = \emptyset$ for all A. To get around this difficulty we need the following preparatory definitions and lemma.

We will construct for any structure $\mathscr{M}$ an adapted structure $\mathscr{M}^a$ in which the axioms hold. To this end, some elements of the universe need to be "relocated" into different types. We will do this using a partial type projection $\pi_{\mathscr{M},T} : M \rightharpoonup TSym$ which is characterised by

$$\begin{aligned} \pi_{\mathscr{M},T}(o) = A \iff & \quad o \in exactInstance_A^{\mathscr{M}} \\ & \text{and } \delta(o) \sqsubseteq A \\ & \text{and } \delta(o) \sqsubseteq B \implies A \sqsubseteq B \text{ for all } B \in TSym\,. \end{aligned} \tag{1.27}$$
eq:pi-def

Funtion π is well-defined: Assume there are two $A, A' \in TSym$ for which the rhs of the above definition is true. By the third condition we have that $A \sqsubseteq A'$ and $A' \sqsubseteq A$, hence, ($\mathscr{T}$ is a poset) $A = A'$.

The idea behind it is that π maps element o to the type it appears to live in ($o \in exactInstance_A^{\mathscr{M}}$) when looking at it from the perspective of $\mathscr{T}$ only. The additional conditions make this well-defined and ensure that domains in the adapted structure remain the same.

def:adapt **Definition 1.47.** Let $(\mathscr{T}, \Sigma)$ by a *Java* signature, $\mathscr{T}^* = (\text{TSym}^*, \sqsubseteq)$ an extension of type system $\mathscr{T}$ and $\mathscr{M} = (M, \delta, I)$ a $(\mathscr{T}^*, \Sigma)$-structure.
The *adapted structure* $\mathscr{M}^a = (M, \delta^a, I^a)$ for $\mathscr{M}$ is the $(\mathscr{T}^a, \Sigma)$ structure with

$$\begin{aligned}
\mathscr{T}^a &= (\text{TSym}^a, \sqsubseteq^a)\\
\text{TSym}^a &= \text{TSym} \cup \{T_o \mid o \notin \text{dom}(\pi)_{\mathscr{M},\mathscr{T}}\} \text{ for new symbols } T_o\\
\sqsubseteq^a &= \text{transitive closure of } \sqsubseteq_{\mathscr{T}} \cup\\
&\qquad \{(T_o, A) \mid o \notin \text{dom}(\pi)_{\mathscr{M},\mathscr{T}}, A \in \text{TSym}, \delta(o) \sqsubseteq A\}\\
\delta^a(o) &= \begin{cases} A & \text{if } \pi_{\mathscr{M},\mathscr{T}}(o) = A\\ T_o & \text{if } o \notin \text{dom}(\pi)_{\mathscr{M},\mathscr{T}} \end{cases}\\
I^a(f) &= I(f) \text{ for symbols } f \in \Sigma\\
I^a(instance_{T_o}) = I^a(exanctInstance_{T_o}) &= \{o\} \text{ for all } o \notin \text{dom}(\pi)_{\mathscr{M},\mathscr{T}}\\
I^a(cast_{T_o})(x) = I^a(default_{T_o}) &= o \text{ for all } o \notin \text{dom}(\pi)_{\mathscr{M},\mathscr{T}}
\end{aligned}$$

The first observation we need to make is that the domains for the types in TSym have not changed, i.e., $A \in \text{TSym} \implies A^{\mathscr{M}^a} = A^{\mathscr{M}}$.

Proof. $\Rightarrow$ Assume $o \in A^{\mathscr{M}^a}$, that is $\delta^a(o) = B \sqsubseteq^a A$ for some $B \in \text{TSym}^a$. If $B = T_o$ then to $T_o \sqsubseteq^a A$ implies by the definition of $\sqsubseteq^a$) (see part (2) of Lemma 1.19) that there must be a type $C \in \text{TSym}$ with $\delta(o) \sqsubseteq C$ and $C \sqsubseteq A$. Hence, also $\delta(o) \sqsubseteq A$, i.e. $o \in A^{\mathscr{M}}$. If $B = \pi_{\mathscr{M},\mathscr{T}}(o)$ then $\delta(o) \sqsubseteq A$ by definition of $\pi_{\mathscr{M},\mathscr{T}}$.
$\Leftarrow$ Assume $o \in A^{\mathscr{M}}$, that is $\delta(o) = B \sqsubseteq A$ for some $B \in \text{TSym}^*$. If $o \in \text{dom}(\pi)_{\mathscr{M},\mathscr{T}}$, then $\pi_{\mathscr{M},\mathscr{T}}(o)$ is the $\sqsubseteq$-smallest supertype in TSym covering B. Hence, $\delta^a(o) = \pi_{\mathscr{M},\mathscr{T}}(o) \sqsubseteq A$. Part (1) of Lemma 1.19 yields $\delta^a(o) \sqsubseteq^a A$ and so $o \in A^{\mathscr{M}^a}$.
On the other hand, if $o \notin \text{dom}(\pi)_{\mathscr{M},\mathscr{T}}$, then $\delta^a(o) = T_o$ and $T_o \sqsubseteq^a B \sqsubseteq^a A$ (by definition of $\sqsubseteq^a$). Again, $\delta^a(o) \sqsubseteq^a A$ and $o \in A^{\mathscr{M}^a}$.
□

This fact is necessary for the construction to be well-defined. If the domains had changed, the definition $I^a(f) = I(f)$ would not make sense.

The next point is that the axioms (1.4.3) and (1.4.3) correspond directly to the second and third condition of the definition (1.27) of $\pi_{\mathscr{M},\mathscr{T}}$ and we get for all o and A:

$$\mathscr{M} \models ThJ_{\mathscr{T}} \implies (\pi_{\mathscr{M},\mathscr{T}}(o) = A \iff o \in exactInstance_A^{\mathscr{M}}) \qquad (1.28)$$ eq:pi-exact

Proof. We show that under assumption of the axioms, the first condition in (1.27) implies the other two. Choose $o \in exactInstance_A^{\mathscr{M}})$ in the following.

Axiom (1.4.3) ensures that $o \in instance_A^{\mathscr{M}}$, and axiom (1.4.3) that $\delta(o) \sqsubseteq A$. (see later more details...)

Assume that the third condition were violated, that is, there is B with $\delta(o) \sqsubseteq B$ and $A \not\sqsubseteq B$. But this allows us to use axiom (1.4.3) to obtain $o \notin instance_B^{\mathscr{M}}$ and (again by axiom (1.4.3)) that $\delta(o) \not\sqsubseteq B$. Contradiction. □

lem:adapt

Proposition 1.48. *Let Σ $\mathscr{T}$, $\mathscr{T}^*$, $\mathscr{T}^a$ be as in Definition 1.47 and $\mathscr{M}$ a $(\mathscr{T}^*,\Sigma)$-structure satisfying $\mathscr{M} \models ThJ_{\mathscr{T}}$.*

1. *The* adapted *structure $\mathscr{M}^a$ of $\mathscr{M}$ is a* Java *structure and*
2. *$\mathscr{M} \models \varphi \iff \mathscr{M}^a \models \varphi$ for all $(\mathscr{T},\Sigma)$-formulas φ.*

Proof.
ad 1. To argue that $\mathscr{M}^a$ is a Java structure we look separately at items 10, 11, and 9 from Figure 1.12, where A ranges of all type symbols in TSym^a.

1. $instance_A^{\mathscr{M}^a} = \{o \in M \mid \delta^a(o) \sqsubseteq^a A\}$
   We have already observed that we have for all $A \in$ TSym
   $$A^{\mathscr{M}^a} = \{o \in M \mid \delta^a(o) \sqsubseteq^a A\} = \{o \in M \mid \delta(o) \sqsubseteq A\} = A^{\mathscr{M}}.$$
   For $A \in$ TSym we know $\mathscr{M} \models \forall x(instance_A(x) \leftrightarrow \exists y(y \doteq x))$ by axiom (1.4.3). Thus $instance_A^{\mathscr{M}} = \{o \in M \mid \delta(o) \sqsubseteq A\}$. By definition of $\mathscr{M}^a$ we als know $instance_A^{\mathscr{M}^a} = instance_A^{\mathscr{M}}$. Together with the initial observation this proves what we want.
   It remains to consider types T_o for $o \notin \text{dom}(\pi)_{\mathscr{M},\mathscr{T}}$. By definition $instance_{T_o}^{\mathscr{M}^a} = \{o\}$.
   $\{o' \in M \mid \delta^a(o') \sqsubseteq^a T_o\} = \{o' \in M \mid \delta^a(o') = T_o\}$ Lemma 1.19(3)
   $= \{o' \in M \mid o' = o\}$ Def. of δ^a
2. $exactInstance_A^{\mathscr{M}^a} = \{o \in M \mid \delta^a(o) = A\}$
   For $A \in TSym$ the valuation $exactInstance_A^{\mathscr{M}^a}$ is the same as $exactInstance_A^{\mathscr{M}}$
   From (1.28), we obtained that $\pi_{\mathscr{M},\mathscr{T}}(o) = A \iff o \in exactInstance_A^{\mathscr{M}^a}$.
   Let $o \in exactInstance_A^{\mathscr{M}^a}$ be given. The implication from right to left gives us that $\pi_{\mathscr{M},\mathscr{T}}(o) = A$ and also $\delta^a(o) = A$ (since $o \in \text{dom}(\pi)_{\mathscr{M},\mathscr{T}}$).
   For the opposite direction, assume now that $\delta^a(o) = A$. Since $A \in$ TSym, it must be that $o \in \text{dom}(\pi)_{\mathscr{M},\mathscr{T}}$ and $\pi_{\mathscr{M},\mathscr{T}}(o) = A$. The implication from right to left entails $o \in exactInstace_A^{\mathscr{M}^a}$.
   Finally, if $T_o \in TSym^a \setminus TSym$ is a type introduced in the adapted type system for $o \notin \text{dom}(\pi)_{\mathscr{M},\mathscr{T}}$, then (by definition of δ^a) o is the only element of that type, and $I^a(exactInstance)$ is defined accordingly.
3. $cast_A^{\mathscr{M}^a}(o) = \begin{cases} o & \text{if } o \in A^{\mathscr{M}^a} \\ default_A^{\mathscr{M}^a} & \text{otherwise} \end{cases}$
   For a type $A \in TSym$, the domain $A^{\mathscr{M}^a} = A^{\mathscr{M}}$ has *not* changed; the definition of δ^a reveals that some elements o may now have a new dynamic type T_o which is a subtype of A, but this does not modify the extension of the type. We can use

axiom (1.4.3) to show that the semantics of the cast is precisely the required. We can use the fact that $A^{\mathscr{M}} = instance_A^{\mathscr{M}^a}$ established in item 1 and leave the proof as an easy excercise.
Again for the types not already present in TSym, the definition of I^a fixes the semantics of the cast symbols correctly.

ad 2.
For the evaluation of a formula, the adaptation $\mathscr{M}^a$ is indistinguishable from the original $\mathscr{M}$. Keep in mind that the syntactical material for φ is that of $(\mathscr{T},\Sigma)$, i.e., neither the types in $\mathrm{TSym}^* \setminus \mathrm{TSym}, \mathrm{TSym}^a \setminus \mathrm{TSym}$ nor the corresponding function and predicate symbols will appear in φ.
Proof by structural induction over quantifications:

- For any quantifier-free φ we have that $\mathscr{M},\beta \models \varphi \iff \mathscr{M}^a,\beta \models \varphi$. This is a direct consequence of the fact that functions and predicates (in Σ) are interpreted identically in $\mathscr{M}$ and $\mathscr{M}^a$.
- Let $\forall x{:}A.\ \varphi$ be a universally quantified formula for $A \in TSym$. We have:

$$\begin{aligned} & \mathscr{M},\beta \models \forall x.\ \varphi \\ \iff\ & \mathscr{M},\beta_x^o \models \varphi \text{ for all } o \in A^{\mathscr{M}} \\ \overset{\text{i.h.}}{\iff}\ & \mathscr{M}^a,\beta_x^o \models \varphi \text{ for all } o \in A^{\mathscr{M}} \\ \overset{(*)}{\iff}\ & \mathscr{M}^a,\beta_x^o \models \varphi \text{ for all } o \in A^{\mathscr{M}^a} \\ \iff\ & \mathscr{M}^a,\beta \models \forall x.\ \varphi \end{aligned}$$

The essential point is $(*)$ relying upon that quantifiers range over the same domains in $\mathscr{M}$ and $\mathscr{M}^a$. We have observed this already in the proof of the first point of this proposition.
The case for the existential quantifier is completely analogous.

lem:AxiomsDiff

Lemma 1.49. *Let $\mathscr{M}$ be a $(\mathscr{T},\Sigma)$-structure satisfying $\mathscr{M} \models ThJ_{\mathscr{T}}$. Then $\mathscr{M} \models \forall x(exactInstance_A(x) \to \neg instance_B(x))$ for all $B \in$* TSym*, $B \neq A$.*

Proof. Consider $o \in M$ with $o \in exactInstance_A^{\mathscr{M}}$. We aim at $o \notin exactInstance_B^{\mathscr{M}}$ for $B \neq A$.

We assume by way of contradiction that $o \in exactInstance_B^{\mathscr{M}}$. The axioms $ThJ_{\mathscr{T}}$ imply in particular $o \in instance_B^{\mathscr{M}}$ and $o \in instance_A^{\mathscr{M}}$. Again using the axioms these entail $\delta(o) \sqsubseteq A$ and $\delta(o) \sqsubseteq B$. If $\delta(o) = B$ we arrive at $B \sqsubset A$ and thus at $o \notin instance_B^{\mathscr{M}}$, contradicting that fact that $\delta(o) = B$ implies $o \in instance_B^{\mathscr{M}}$.

If $\delta(o) \sqsubset B$ we obtain $o \notin instance_{\delta(o)}^{\mathscr{M}}$ contradicting the fact that the axioms imply $o \in instance_{\delta o}^{\mathscr{M}}$. □

The next lemma claims that ThJ is a complete axiomatization of *Java* structures.

lem:JavaCompleteA

Lemma 1.50. *Let $(\mathscr{T},\Sigma)$ be a* Java *signature and ϕ a $(\mathscr{T},\Sigma)$ sentence. Then*

*$ThJ_{\mathscr{T}} \models \phi \Leftrightarrow$ for all extensions $\mathscr{T}^* \sqsupseteq \mathscr{T}$ and all $(\mathscr{T}^*,\Sigma)$-*Java*-structures $\mathscr{M}$ $\mathscr{M} \models \phi$*

Proof. For the implication $\Rightarrow$ from left to right we assume $ThJ_{\mathcal{T}} \models \phi$ and fix an extension hierachy $\mathcal{T}^* \sqsupseteq \mathcal{T}$ and a $(\mathcal{T}^*, \Sigma)$-*Java* structure $\mathcal{M}$ with the aim of showing $\mathcal{M} \models \phi$. We will succeed if we can show $\mathcal{M} \models ThJ_{\mathcal{T}}$, i.e. if we can show that the axioms from Figure 1.8 are true assuming that $\mathcal{M}$ is definied according to Figure 1.12. Axioms $(Ax-I)$ $(Ax-C)$ $(Ax-L)$ are explicit definitions while $(Ax-E_1)$ $(Ax-E_2)$ are direct consequences of the semantics of $exactInstance_A$. For the reverse implication, $\Leftarrow$, we assume the lefthand condition and fix an extension $\mathcal{T}^* \sqsupseteq \mathcal{T}$ and a $(\mathcal{T}^*, \Sigma)$-structure $\mathcal{M}$ with $\mathcal{M} \models ThJ_{\mathcal{T}}$. We want to arrive at $\mathcal{M} \models \phi$. Let $\mathcal{M}^a$ be the adapted structure for $\mathcal{M}$ as in Definition 1.47. By the first part of Proposition 1.48 we know that $\mathcal{M}^a$ is a *Java* structure. By assumption this implies $\mathcal{M}^a \models \phi$. By the second part of Proposition 1.48 we get $\mathcal{M} \models \phi$. $\square$

End of Digression on the Proof of Lemma 1.8

1.4.8 Digression on Soundness of Array Theory

lem:ArraySound

Lemma 1.51. *The taclets in Figure 1.6 are sound with respect to the semantics.*

Proof

The three taclets in Figure 1.6 are rewrite rules. So we need to show that for any JFOL structure $\mathcal{M}$ the interpretation in $\mathcal{M}$ of the term below the horizontal line equals the interpretation in $\mathcal{M}$ of the term above the horizontal line equals

selectOfStore

Fix an arbitrary JFOL structure $\mathcal{M}$ and elements $h \in D^{Heap}$, $o, o_2 \in D^{Object}$, $f, f_2 \in D^{Field}$, and $x \in D^{Any}$. We compute $select_A(store(h,o,f,x),o_2,f_2)^{\mathcal{M}}$ as follows. By item 3 in Figure 1.12 $h^* = store(h,o,f,x)^{\mathcal{M}}$ is given by
$$h^*(o',f') = \begin{cases} x & \text{if } o' = o, f = f' \text{ and } f \neq created^{\mathcal{M}} \\ h(o',f') & \text{otherwise} \end{cases}$$
By item 2 in Figure 1.12 $select_A(h^*,o_2,f_2)^{\mathcal{M}} = cast_A^{\mathcal{M}}(h^*(o_2,f_2))$. According to the definition of h^* there are 2 cases to be considered.
Case 1 $o' = o_2$ and $f' = f_2$ and $f' \neq created^{\mathcal{M}}$
In this case $h^*(o_2,f_2) = x$ and $select_A(h^*,o_2,f_2)^{\mathcal{M}} = cast_A^{\mathcal{M}}(x)$.
Case 2 otherwise
In this case $h^*(o_2,f_2) = h(o_2,f_2)$ and $select_A(h^*,o_2,f_2)^{\mathcal{M}} = cast_A^{\mathcal{M}}(h(o_2,f_2)) = select_A(h(o_2,f_2))^{\mathcal{M}}$.
This coincides with the evaluation in $\mathcal{M}$ of the term to the right of $\rightsquigarrow$

$$\text{if } o \doteq o_2 \land f \doteq f_2 \land f \neq \mathit{created} \text{ then } \mathit{cast}_A(x) \text{ else } \mathit{select}_A(h, o_2, f_2)$$

selectOfCreate

Fix an arbitrary JFOL structure $\mathscr{M}$ and elements $h \in D^{Heap}$, $o, o_2 \in D^{Object}$, $f \in D^{Field}$. We compute $\mathit{select}_A(\mathit{create}(h,o), o2, f)^{\mathscr{M}}$ as follows. By item 4 in Figure 1.12 the heap $h^* = \mathit{create}(h,o)^{\mathscr{M}}$ is given by

$$h^*(o', f) = \begin{cases} tt & \text{if } o' = o, o \neq \mathit{null} \text{ and } f = \mathit{created}^{\mathscr{M}} \\ h(o', f) & \text{otherwise} \end{cases}$$

Thus our task is reduced to the computation of $\mathit{select}_A(h^*, o2, f)^{\mathscr{M}}$, which by item 3 equals $h^*(o2, f)$.
This coincides with the evaluation in $\mathscr{M}$ of the term to the right of $\rightsquigarrow$

$$\text{if } o \doteq o2 \land o \dot{\neq} \mathit{null} \land f \doteq \mathit{created} \text{ then } \mathit{cast}_A(\mathit{TRUE}) \text{ else } \mathit{select}_A(h, o2, f)$$

selectOfAnon

Fix an arbitrary JFOL structure $\mathscr{M}$ and elements $h, h' \in D^{Heap}$, $o \in D^{Object}$, $f \in D^{Field}$, and $s \in d^{LocSet}$. We compute $\mathit{select}_A(\mathit{anon}(h,s,h'), o, f)^{\mathscr{M}}$ as follows. By item 26 in Figure 1.12 the heap $h^* = \mathit{anon}(h,s,h')^{\mathscr{M}}$ is given by

$$h^*(o, f) = \begin{cases} h'(o,f) \text{ if } (o,f) \in s \text{ and } f \neq \mathit{created}^{\mathscr{M}}, \text{ or} \\ \qquad\qquad (o,f) \in \mathit{unusedLocs}^{\mathscr{M}}(h) \\ h(o,f) \text{ otherwise} \end{cases}$$

Thus our task is reduced to the computation of $\mathit{select}_A(h^*, o, f)^{\mathscr{M}}$, which by item 3 equals $h^*(o, f)$.
This coincides with the evaluation in $\mathscr{M}$ of the term to the right of $\rightsquigarrow$

$$\begin{array}{l} \text{if}(\varepsilon(o, f, s) \land f \dot{\neq} \mathit{created}) \lor \varepsilon(o, f, \mathit{unusedLocs}(h)) \\ \text{then } \mathit{select}_A(h', o, f) \text{ else } \mathit{select}_A(h, o, f) \end{array}$$

1.5 Finite Sequences

sec05:FinSeq

This section develops and explains the theory T_{seq} of finite sequences. By *Seq* we denote the type of finite sequences. The vocabulary Σ_{seq} of the theory is listed in Figure 1.14. We will start with a simple core theory CoT_{seq} and incrementally enrich it via definitional extensions. Typing of a function symbol f is given as $f : A_1 \times \ldots \times A_n \to R$ with argument types A_i and result type R, typing of a predicate symbol p as $p(A_1 \times \ldots \times A_n)$.

Core Theory
$A{::}seqGet : Seq \times int \to A$ for any type $A \sqsubseteq Any$
$seqGetOutside : Any$
$seqLen : Seq \to int$

Variable Binder
$seqDef : int \times int \times Seq \to Seq$

Definitional Extension
$seqDepth : Seq \to int$
$seqEmpty : Seq$
$seqSingleton : Any \to Seq$
$seqConcat : Seq \times Seq \to Seq$
$seqSub : Seq \times int \times int \to Seq$
$seqReverse : Seq \to Seq$
$seqIndexOf : Seq \times Any \to int$
$seqNPerm(Seq)$
$seqPerm(Seq, Seq)$
$seqSwap : Seq \times int \times int \to Seq$
$seqRemove : Seq \times int \to Seq$
$seqNPermInv : Seq \to Seq$

Fig. 1.14 The vocabulary Σ_{seq} of the theory T_{seq} of finite sequences

fig:SigmaSeq

Our notion of a sequence is rather liberal, e.g., $\langle 5,6,7,8\rangle$ is a sequence, in fact a sequence of integers. But the heterogeneous, nested list $\langle 0, \langle \emptyset, seqEmpty, null\rangle, true\rangle$ is also allowed.

The semantics of the symbols of the core theory will be given in Definition 1.54. We provide here a first informal account of the intended meaning. The function value $seqLen(s)$ is the length of list s. Since for heterogeneous lists there is no way the type of an entry can be recovered from the type of the list, we provide a family of access functions $seqGet_A$ that yields the cast to type A of the i-th entry in list s. (The concrete, ASCII syntax is $A{::}seqGet$, but we stick here with the slightly shorter notation $seqGet_A$.) The constant *seqGetOutside* is an arbitrary element of the top type *Any*. It is, e.g., used as the value of any attempt to access a sequence outside its range. *seqDef* is a variable binder symbol, check Section 1.3.1 for explanation. Its precise semantics is given in Definition 1.53 below. The reader may get a first intuition from the simple example $seqDef\{u\}(1,5,u^2)$ that represents the sequence $\langle 1,4,9,16\rangle$. We will comment on the symbols in the definitional extension, when we are finished with the core theory following page 63.

lenNonNegative
$\forall Seq\ s; (0 \leq seqLen(s))$

equalityToSeqGetAndSeqLen
$\forall Seq\ s_1, s_2; (s_1 \doteq s_2 \leftrightarrow seqLen(s_1) \doteq seqLen(s_2) \land$
$\quad \forall int\ i; (0 \leq i < seqLen(s_1) \to seqGetAny(s_1, i) \doteq seqGet_{Any}(s_2, i)))$

getOfSeqDef
$\forall int\ i, ri, le; \forall Any\ \bar{x}; ($
$\quad ((0 \leq i \land i < ri - le) \to seqGet_A(seqDef\{u\}(le, ri, t), i) \doteq cast_A(t\{(le+i)/u\})) \land$
$\quad (\neg(0 \leq i \land i < ri - le) \to seqGet_A(seqDef\{u\}(le, ri, t), i) \doteq cast_A(seqGetOutside)))$

lenOfSeqDef
$\forall int\ ri, le; ((le < ri \to seqLen(seqDef\{u\}(le, ri, t)) \doteq ri - le) \land$
$\quad (ri \leq le \to seqLen(seqDef\{u\}(le, ri, t)) \doteq 0))$

Fig. 1.15 Axioms of the core theory CoT_{seq} (in mathematical notation)

fig:SeqCoreAxiomsMath

The axioms of the core theory CoT_{seq} are shown in Figure 1.15 in mathematical notation together with the names of the corresponding taclets. In getOfSeqDef the quantifier $\forall Any\ \bar{x}$ binds the variables that may occur in term t.

Definition 1.52 below extends the semantics of type domains given in Figure 1.11 on page 51. More precisely, the definition gives the construction to obtain D^{Seq} when all other type domains are fixed.

defi:SeqDomain

Definition 1.52 (The type domain D^{Seq}). The type domain D^{Seq} is defined via the following induction:

$$D^{Seq} \quad := \quad \bigcup_{n \geq 0} D^n_{\text{Seq}}$$

where

$$\begin{aligned} U &= D^{Any} \setminus D^{Seq} \\ D^0_{\text{Seq}} &= \{\langle\rangle\} \\ D^{n+1}_{\text{Seq}} &= \{\langle a_0, \ldots, a_{k-1}\rangle \mid k \in \mathbb{N} \text{ and } a_i \in D^n_{\text{Seq}} \cup U, 0 \leq i < k\} \quad \text{for } n \geq 0 \end{aligned}$$

The type domain for *Seq* being fixed we now may deliver on the forward reference after Definition 1.35 and define precisely the meaning of the variable binder symbol $seqDef\{iv\}(le, ri, e)$ in the JFOL structure $\mathcal{M}$. As already done in Section 1.4.6 we will use the notation $t^{\mathcal{M},\beta}$ for term evaluation instead of $\text{val}_{\mathcal{M},\beta}(t)$. We further will suppress β and write $t^{\mathcal{M}}$ if it is not needed or not relevant.

defi:seqDefSemantics

Definition 1.53.

$$seqDef\{iv\}(le, ri, e)^{\mathcal{M},\beta} = \begin{cases} \langle a_0, \ldots a_{k-1}\rangle & \text{if } (ri - le)^{\mathcal{M},\beta} = k > 0 \text{ and } a_i = e^{\mathcal{M},\beta_i} \\ & \text{with } \beta_i = \beta[le + i/iv] \text{ and all } 0 \leq i < k \\ \langle\rangle & \text{otherwise} \end{cases}$$

Remember, that $\beta[le + i/iv]$ is the variable assignment that coincides with β expect for the argument iv where it takes the value $le + i$.

The core vocabulary of CoT_{seq} is interpreted as follows:

defi:SeqModels

Definition 1.54.

item:SeqGet

1. $seqGet_A^{\mathcal{M}}(\langle a_0,\ldots,a_{n-1}\rangle, i) = \begin{cases} cast_A^{\mathcal{M}}(a_i) & \text{if } 0 \le i < n \\ cast_A^{\mathcal{M}}(seqGetOutside^{\mathcal{M}}) & \text{otherwise} \end{cases}$

item:SeqLen

2. $seqLen^{\mathcal{M}}(\langle a_0,\ldots,a_{n-1}\rangle) = n$

item:seqGetOutside

3. $seqGetOutside^{\mathcal{M}} \in D^{Any}$ arbitrary.

To have a name for it we might call a structure $\mathcal{M}$ in the vocabulary Σ_J (see Figure 1.4) plus the core vocabulary of finite sequences a *CoreSeq* structure, if its restriction to the JFOL vocabulary is a JFOL structure as defined in Section 1.4.6 and, in addition $\mathcal{M}$ satisfies Definition 1.54. We observe, that the expansion of a JFOL structure $\mathcal{M}_0$ to a *CoreSeq* structure is uniquely determined once an interpretation $seqGetOutside^{\mathcal{M}_0}$ is chosen.

thm:CoTseqConsistent

Theorem 1.55. *The theory CoT_{seq} is consistent.*

Proof. It is easily checked that the axioms in Figure 1.15 are true in all *CoreSeq* structures. The explicit construction guarantees that there is at least one *CoreSeq* structure.

1.5.1 Digression on Core Consistency

Detailed proof of Theorem 1.55.

Proof. We consider the axioms from Figure 1.15 one by one.
$lenNonNegative$
$\forall Seq\ s(0 \le seqLen(s))$
This is obvious by item 2 in Definition 1.54.

$equalityToSeqGetAndSeqLen$
$\forall Seq\ s_1, s_2(s_1 \doteq s_2 \leftrightarrow$
$\quad seqLen(s_1) \doteq seqLen(s_2) \land \forall Int\ i(0 \le i < seqLen(s_1) \to s_1[i] \doteq s_2[i]))$
Obvious, by definition of D^{Seq}.

$getOfSeqDef$
$\forall Int\ i, ri, le(((0 \le i \land i < ri - le) \to$
$seqGet_A(seqDef\{u\}(le, ri, t), i) \doteq cast_A(t\{(le+i)/u\}))$
$\land$
$(\neg(0 \le i \land i < ri - le) \to$
$\quad seqGet_A(seqDef\{u\}(le, ri, t), i) \doteq cast_A(seqGetOutside)))$
Let t a term of type Seq and β a variable assignment with $\beta(i) = n_i$, $\beta(ri) = n_{ri}$, $\beta(le) = n_{le}$.
Case $\mathbf{0 \le n_i \land n_i < n_{ri} - n_{le}}$
By Definitions 1.52 and 1.54
$seqGet_A(seqDef\{u\}(le, ri, t), i)^{\mathcal{M},\beta} = cast_A^{\mathcal{M}}(t^{\mathcal{M},\beta[n_{le}+n_i/u]})$

The substitution lemma yields $t^{\mathcal{M},\beta[n_{le}+n_i/u]} = t\{(le+i)/u\}^{\mathcal{M},\beta}$, as desired.
Case $(\mathbf{n_{ri}} - \mathbf{n_{le}}) \geq \mathbf{0} \wedge \neg(\mathbf{0} \leq \mathbf{n_i} \wedge \mathbf{n_i} < \mathbf{n_{ri}} - \mathbf{n_{le}})$
Definition 1.54 immediately gives:
$seqGet_A(seqDef\{u\}(le,ri,t),i)^{\mathcal{M},\beta} = cast_A^{\mathcal{M}}(seqGetOutside^{\mathcal{M}})$.
Case $(\mathbf{n_{ri}} - \mathbf{n_{le}}) < \mathbf{0}$
We note first that the case assumption implies $\neg(0 \leq n_i \wedge n_i < n_{ri} - n_{le})$. By Definition 1.53 $seqDef\{u\}(le,ri,t)^{\mathcal{M},\beta} = \langle\rangle$. Now, Definition 1.54 gives:
$seqGet_A(seqDef\{u\}(le,ri,t),i)^{\mathcal{M},\beta} = cast_A^{\mathcal{M}}(seqGetOutside^{\mathcal{M}})$.

l$enOfSeqDef$
$\forall Int\ ri, le($
$(le < ri \rightarrow seqLen((seqDef\{u\}(le,ri,t)) \doteq ri - li)$
$\wedge$
$(ri \leq le \rightarrow seqLen(seqDef\{u\}(le,ri,t)) \doteq 0))$
Let t and β be as in the previous item.
Case $(\mathbf{n_{ri}} - \mathbf{n_{le}}) \geq \mathbf{0}$
By Definition 1.53 $seqDef\{u\}(le,ri,t)^{\mathcal{M},\beta}$ is of the form $\langle a_0,\ldots,a_{k-1}\rangle$ with $k = n_{re} - n_{le}$. Thus Definition 1.54 yields $seqLen(seqDef\{u\}(le,ri,t))^{\mathcal{M},\beta} = k$.
Case $(\mathbf{n_{ri}} - \mathbf{n_{le}}) < \mathbf{0}$
By Definition 1.53 $seqDef\{u\}(le,ri,t)^{\mathcal{M},\beta} = \langle\rangle$. □

End of Digression on Core Consistency

$$\begin{array}{l}
\forall Seq\ s; (\forall int\ i; ((0 \leq i < seqLen(s) \rightarrow \neg instance_{Seq}(seqGet_{Any}(s,i))) \rightarrow seqDepth(s) \doteq 0) \wedge \\
\forall Seq\ s; (\forall int\ i; ((0 \leq i < seqLen(s) \wedge instance_{Seq}(seqGet_{Any}(s,i))) \rightarrow \\
\qquad\qquad seqDepth(s) > seqDepth(seqGet_{Seq}(s,i))) \wedge \\
\forall Seq\ s; (\exists int\ i; (0 \leq i < seqLen(s) \wedge instance_{Seq}(seqGet_{Any}(s,i))) \rightarrow \\
\qquad\quad \exists int\ i; (0 \leq i < seqLen(s) \wedge instance_{Seq}(seqGet_{Any}(s,i)) \wedge \\
\qquad\qquad seqDepth(s) \doteq seqDepth(seqGet_{Seq}(s,i)) + 1)
\end{array}$$

Fig. 1.16 Definition of *seqDepth* fig:defSeqDepth

We observe that $seqDepth(s)$ as defined in Figure 1.16 equals the recursive definition

$$seqDepth(s) = \max\{seqDepth(seqGet_{Seq}(s,i)) \mid 0 \leq i < seqLen(s) \wedge instance_{Seq}(seqGet_{Seq}(s,i))\}$$

with the understanding that the maximum of the empty set is 0. Since we have not introduced the maximum operator we had to resort to the formula given above. The function *seqDepth* is foremost of theoretical interest and at the moment of this writing not realized in the KeY system. $seqDepth(s)$ is an integer denoting the *nesting depth* of sequence s. If s has no entries that are themselves sequences then $seqDepth(s) \doteq 0$. For a sequence s_{int} of sequences of integers we would have $seqDepth(s_{\text{int}}) \doteq 0$.

In Figure 1.17 the mathematical formulas defining the remaining noncore vocabulary are accompanied by the names of the corresponding taclets. A few explaining comments will help the reader to grasp their meaning. The subsequence $seqSub(s,i,j)$ from i to j of sequence s includes the i-th entry, but excludes the j-th entry. In the case $\neg(i < j)$ it will be the empty sequence, this is a consequence of the semantics of $seqDef$. The term $seqIndexOf(s,t)$ denotes the least index n such that $seqGet_{Any}(s,n) \doteq t$ if there is one, and is undefined otherwise. See Section 1.3.2 on how undefinedness is handled in our logic. A sequence s satisfies the predicate $seqNPerm(s)$ if it is a permutation of the integers $\{0,\ldots,seqLen(s)-1\}$. The binary predicate $seqPerm(s_1,s_s)$ is true if s_2 is a permutation of s_1. Thus $seqNPerm(\langle 5,4,0,2,3,1\rangle)$ and $seqPerm(\langle a,b,c\rangle,\langle b,a,c\rangle)$ are true.

Careful observation reveals that the interpretation of the vocabulary outside the core vocabulary is uniquely determined by the definitions in Figures 1.16 and 1.17.

We establish the following notation:

defi:Tseq

Definition 1.56. By T_{seq} we denote the theory given by the core axioms CoT_{seq} plus the definitions from Figures 1.16 and 1.17.

On the semantic side we call a structure $\mathscr{M}$ in the vocabulary Σ_J plus Σ_{Seq} a *Seq* structure if the restriction of $\mathscr{M}$ to Σ_J is a JFOL structure and $\mathscr{M}$ satisfies Definitions 1.54 and 1.17.

thm:seqTheory

Theorem 1.57. *The theory T_{Seq} is consistent.*

Proof. The consistency of T_{Seq} follows from the consistency of CoT_{Seq} since it is a definitional extension.

1.5.2 Digression on Definitional Extensions

In this digressive subsection a proof of Theorem 1.57 will eventually be given in Lemma 1.63. The proof plan is like this: The core theory CoT_{Seq} is consistent. Every definitional extension of a consistent theory is also consistent. T_{Seq} is a definitional extension of CoT_{Seq}.

Monk76,Shoenfield67,EbbinghausFlumThomas07

The reference Monk [1976], Shoenfield [1967], Ebbinghaus et al. [2007] cited above may not be accessible to everyone, and are also quite dense. We will provide in this subsection the full theoretical background on this topic.

We need some terminology first.

defi:CExt

Definition 1.58 (Conservative Extension). Let $\Sigma_0 \subseteq \Sigma_1$ be signatures, and T_i set of sentences in Fml_{Σ_i}.
T_1 is called a *conservative extension* of T_0 if for all sentences $\phi \in Fml_{\Sigma_0}$:

$$T_0 \vdash \phi \Leftrightarrow T_1 \vdash \phi$$

defOfEmpty
$seqEmpty \doteq seqDef\{iv\}(0,0,x)$
x is an arbitrary term of type *Any* not containing the variable *iv*.

defOfSeqSingleton
$\forall Any\ x; (seqSingleton(x) \doteq seqDef\{iv\}(0,1,x))$

defOfSeqConcat
$$\begin{aligned}&\forall Seq\ s_1, s_2; (seqConcat(s_1,s_2) \doteq\\ &\quad seqDef\{iv\}(0, seqLen(s_1) + seqLen(s_2),\ \text{if } iv < seqLen(s_1)\\ &\qquad \text{then } seqGet_{Any}(s_1, iv)\\ &\qquad \text{else } seqGet_{Any}(s_2, iv - seqLen(s_1)))))\end{aligned}$$

defOfSeqSub
$\forall Seq\ s; \forall int\ i,j; (seqSub(s,i,j) \doteq seqDef\{iv\}(i,j,seqGet_{Any}(s,iv)))$

defOfSeqReverse
$\forall Seq\ s; (seqReverse(s) \doteq seqDef\{iv\}(0, seqLen(s), seqGet_{Any}(s, seqLen(s) - iv - 1)))$

seqIndexOf
$$\begin{aligned}&\forall Seq\ s; \forall Any\ t; \forall int\ n; (0 \leq n < seqLen(s) \land seqGet_{Any}(s,n) \doteq t \land\\ &\quad \forall int\ m; (0 \leq m < n \rightarrow seqGet_{Any}(s,m) \neq t)\\ &\quad \rightarrow seqIndexOf(s,t) \doteq n)\end{aligned}$$

seqNPermDefReplace
$$\begin{aligned}&\forall Seq\ s; (seqNPerm(s) \leftrightarrow\\ &\quad \forall int\ i; (0 \leq i < seqLen(s) \rightarrow \exists int\ j; (0 \leq j < seqLen(s) \land seqGet_{int}(s,j) \doteq i)))\end{aligned}$$

seqPermDef
$$\begin{aligned}&\forall Seq\ s_1, s_2; (seqPerm(s_1,s_2) \leftrightarrow seqLen(s_1) \doteq seqLen(s_2) \land\\ &\quad \exists Seq\ s; (seqLen(s) \doteq seqLen(s_1) \land seqNPerm(s) \land\\ &\quad \forall int\ i; (0 \leq i < seqLen(s) \rightarrow\\ &\qquad seqGet_{Any}(s_1,i) \doteq seqGet_{Any}(s_2, seqGet_{int}(s,i)))))\end{aligned}$$

defOfSeqSwap
$$\begin{aligned}&\forall Seq\ s; \forall int\ i,j; (seqSwap(s,i,j) \doteq\\ &\quad seqDef\{iv\}(0, seqLen(s),\ \text{if } \neg(0 \leq i < seqLen(s) \land 0 \leq j < seqLen(s))\\ &\qquad \text{then } seqGet_{Any}(s,iv)\\ &\qquad \text{else if } iv \doteq i\\ &\qquad\quad \text{then } seqGet_{Any}(s,j)\\ &\qquad\quad \text{else if } iv \doteq j\\ &\qquad\qquad \text{then } seqGet_{Any}(s,i)\\ &\qquad\qquad \text{else } seqGet_{Any}(s,iv)))\end{aligned}$$

defOfSeqRemove
$$\begin{aligned}&\forall Seq\ s; \forall int\ i; (seqRemove(s,i) \doteq \text{if } i < 0 \lor seqLen(s) \leq i\\ &\quad \text{then } s\\ &\quad \text{else} seqDef\{iv\}(0, seqLen(s) - 1,\ \text{if } iv < i\\ &\qquad \text{then } seqGet_{Any}(s,iv)\\ &\qquad \text{else } seqGet_{Any}(s,iv+1)))\end{aligned}$$

defOfSeqNPermInv
$\forall Seq\ s; (seqNPermInv(s) \doteq\ seqDef\{iv\}(0, seqLen(s), seqIndexOf(s,iv)))$

Fig. 1.17 Definition for noncore vocabulary in mathematical notation fig:SeqAxiomsMath

Note, if T_0 is consistent and T_1 is a conservative extension of T_0 then T_1 is also consistent.

defi:SemanticCExt

Definition 1.59 (Semantic Conservative Extension). Let $\Sigma_0 \subseteq \Sigma_1$ be signatures, and T_i sets of sentences in Fml_{Σ_i}.
T_1 is called a *semantic conservative extension* of T_0 if

1. for all Σ_1-structures $\mathscr{M}_1$ with $\mathscr{M}_1 \models T_1$ the restriction $\mathscr{M}_0$ of $\mathscr{M}_1$ to Σ_0 is a model of T_0, in symbols
$$\mathscr{M}_1 \models T_1 \Rightarrow (\mathscr{M}_1 \restriction \Sigma_0) \models T_0$$
2. for every Σ_0-structure $\mathscr{M}_0$ with $\mathscr{M}_0 \models T_0$ there is a Σ_1-expansion $\mathscr{M}_1$ of $\mathscr{M}_0$ with $\mathscr{M}_1 \models T_1$.

Note, in case $T_0 \subseteq T_1$ is true, which is the most typical case, but not required in Definitions 1.58 and 1.59, then item 1 of the preceeding definition is automatically true.

lemma:SemCExt->CExt

Lemma 1.60. *Let $\Sigma_0 \subseteq \Sigma_1$ be signatures, and T_i sets of sentences in Fml_{Σ_i}.*
If T_1 is a semantic conservative extension of T_0
then T_1 is also a conservative extension of T_0

Proof. Let ϕ be a sentence in Fml_{Σ_0} with $T_0 \vdash \phi$. Let $\mathscr{M}_1$ be an arbitrary Σ_1-structure. By assumption $(\mathscr{M}_1 \restriction \Sigma_0) \models T_0$. Thus we also have $(\mathscr{M}_1 \restriction \Sigma_0) \models \phi$. By the coincidence lemma we also have $\mathscr{M}_1 \models \phi$. In total we have shown $T_1 \vdash \phi$.
Now, assume $T_1 \vdash \phi$. If $\mathscr{M}_0$ is an arbitrary Σ_0-structure there is by the assumption an expansion of $\mathscr{M}_0$ to a Σ_1-structure $\mathscr{M}_1$. From $T_1 \vdash \phi$ we thus get $\mathscr{M}_1 \models \phi$. The coincidence lemma tells us again that also $\mathscr{M}_0 \models \phi$. In total we arrive at $T_o \vdash \phi$.

lem:DefExt

Lemma 1.61 (Extension by Definition). *Let $\Sigma_0 \subseteq \Sigma_1$ be signatures, $T_0 \subseteq T_1$ sets of sentences in Fml_{Σ_0} respectively in Fml_{Σ_1}. Further assume that all sentences in $T_1 \setminus T_0$ are of the form*

$$\begin{array}{ll} \forall \bar{x}(f(\bar{x}) \doteq t) & f \in \Sigma_1 \subseteq \Sigma_0 \ t \text{ a term in } \Sigma_0 \\ \forall \bar{x}(p(\bar{x}) \leftrightarrow \phi(\bar{x}) & p \in \Sigma_1 \subseteq \Sigma_0 \ \phi \text{ a formula in } \Sigma_0 \end{array}$$

Then T_1 is a semantic conservative extension of T_0.

Proof. If $\mathscr{M}_0$ is a Σ_0-model of T_0 we obtain an Σ_1-expansion $\mathscr{M}_1$ by simply setting
$$f^{\mathscr{M}_1}(\bar{a}) = t^{\mathscr{M}_0}(\bar{a})$$
and
$$p^{\mathscr{M}_1}(\bar{a}) \Leftrightarrow \mathscr{M}_0 \models \phi[\bar{a}]$$

In the situation of Lemma 1.61 T_1 is called an extension by definitions of T_0. We tacitly assume – of course – that T_1 contains only one definition for each new function or relation symbol.

This lemma covers the extensions of CoT_{seq} by all the defining taclets in Figure 1.17 from d$efOfEmpty$ to d$efOfSeqNPermInv$ except s$eqIndexOf$. That the definition of s$eqIndexOf$ leads to a conservative extension is the content of the following lemma.

lem:UniqueDefExt

Lemma 1.62 (Unique Conditional Extension by Definition). *Let $\Sigma_0 \subseteq \Sigma_1$ be signatures, $T_0 \subseteq T_1$ sets of sentences in Fml_{Σ_0} respectively in Fml_{Σ_1}. Further assume that all sentences in $T_1 \setminus T_0$ are of the form*

$$\begin{array}{ll} \forall \bar{x} \forall y(\psi \to f(\bar{x}) \doteq y) & f \in \Sigma_1 \subseteq \Sigma_0 \\ & \psi \textit{ a formla in } \Sigma_0 \\ \textit{such that} & \\ T_0 \vdash \forall \bar{x} \forall y, y'(\psi \wedge \psi\{y'/y\} \to y \doteq y') & \end{array}$$

Then T_1 is a semantic conservative extension of T_0.

Proof. We obtain a Σ_1 extension $\mathscr{M}_1$ of a Σ_0 model $\mathscr{M}_0$ of T_0 by defining

$$f^{\mathscr{M}_1}(\bar{a}) = \begin{cases} b & \text{if } \mathscr{M}_0 \models \psi[\bar{a}, b] \\ \text{arbitrary} & \text{otherwise} \end{cases}$$

Since for any $\bar{a}$ there can be at most one b satisfying $\mathscr{M}_0 \models \psi[\bar{a}, b]$ this is a sound definition.

lem:seqCore1

Lemma 1.63. *T_{seq} is a conservative extension of CoT_{seq}, and thus in particular consistent.*

Proof. Inspection of the axioms shows that they are all of the syntactic form required by Lemma 1.61, except for the definition of $seqIndexOf$ which follows that pattern offered in Lemma 1.62. The formula to be proved in seqCore is in this case

$$\begin{array}{l} \forall s \forall a \forall j, j'(\\ (0 \leq j \wedge j < seqLen(s) \wedge s[j] \doteq a \wedge \forall k(0 \leq k \wedge k < j \to s[k] \neq a)) \wedge \\ (0 \leq j' \wedge j' < seqLen(s) \wedge s[j'] \doteq a \wedge \forall k(0 \leq k \wedge k < j' \to s[k] \neq a)) \\ \to j' \doteq j) \end{array}$$

This can easily seen to be true.

Suppose we had instead of the partial function $seqIndexOf$ introduced a weaker version $wSeqIndexOf$ by the definition:

$$\begin{array}{l} \text{w}SeqIndexOfFalse \\ \forall Seq\ s \forall Any\ t \forall Int\ n(\ (0 \leq n < seqLen(s) \wedge seqGet_{Any}(s,n) \doteq t) \\ \qquad\qquad \to wSeqIndexOf(s,t)) \doteq n) \end{array}$$

The function $wSeqIndexOf(s,t)$ picks any index n with $seqGet_{Any}(s,n) \doteq t$ if there exists one, instead of the smallest index. This would lead to an inconsistent theory. For $s = \langle 5, 5 \rangle$ one could derive $0 \doteq wSeqIndexOf(s,5) \doteq 1$.

But, the following defintion would be okay:

$$\begin{array}{ll}\mathsf{w}\mathit{SeqIndexOf}1 & \\ \forall \mathit{Seq}\ s \forall \mathit{Any}\ t & \forall \mathit{Int}\ n((0 \leq n < \mathit{seqLen}(s) \wedge \mathit{seqGet}_{\mathit{Any}}(s,n) \doteq t) \to \\ & (0 \leq \mathit{wSeqIndexOf}(s,t) < \mathit{seqLen}(s) \wedge \\ & \quad \mathit{seqGet}_{\mathit{Any}}(s, \mathit{wSeqIndexOf}(s,t)) \doteq t))\end{array}$$

This is logically equivalent to

$$\begin{array}{ll}\mathsf{w}\mathit{SeqIndexOf} & \\ \forall \mathit{Seq}\ s \forall \mathit{Any}\ t & (\exists \mathit{Int}\ n((0 \leq n < \mathit{seqLen}(s) \wedge \mathit{seqGet}_{\mathit{Any}}(s,n) \doteq t))) \to \\ & (0 \leq \mathit{wSeqIndexOf}(s,t) < \mathit{seqLen}(s) \wedge \\ & \quad \mathit{seqGet}_{\mathit{Any}}(s, \mathit{wSeqIndexOf}(s,t)) \doteq t)\end{array}$$

Now, it is obvious that this is just an instance if the classical Skolem extension lemma which we repeat here for the reader convenience.

lem:skolemExt

Lemma 1.64. *Let T_0 be a Σ_0-theory, $\Sigma_1 = \Sigma_0 \cup \{f\}$ where f is a new n-place function symbol and let T_1 be obtained from T_0 by adding an axiom of the following form*

$$\forall \bar{x}(\exists y(\phi) \to \phi\{f(\bar{x})/y\})$$

then T_1 is a conservative extension of T_0.
Here $\bar{x}$ is a tupel of variables of the same length n as the argument tupel of f and, as before $\phi\{f(\bar{x})/y\}$ denotes the formula arising from ϕ by replacing all free occurrences of y by $f(\bar{x})$.

Proof. We show that T_1 is a semantic conservative extension of T_0. Let $\mathscr{M}_0$ be a model of T_0. The structure $\mathscr{M}_1$ coincides with $\mathscr{M}_0$ for all Σ_0-sybols. We define an interpretation of the symbol f as follows

$$f^{\mathscr{M}_1}(\bar{a}) = \begin{cases} b & \text{if } \mathscr{M}_0 \models \exists y(\phi)[\bar{a}] \\ & \text{then pick } b \text{ with } \mathscr{M}_0 \models \phi[\bar{a}, b] \\ \text{arbitrary} & \text{otherwise} \end{cases}$$

Technical note, we use $\mathscr{M}_0 \models \phi[\bar{a}, b]$ as a shorthand for $(\mathscr{M}_0, \beta) \models \phi$ with the variable assignment defined by $\beta(x_i) = a_i$ for $0 \leq i < n$ and $\beta(y) = b$.

Obviously, $\mathscr{M}_1 \models \forall \bar{x}(\exists y(\phi) \to \phi[f(\bar{x})/y])$

1.5.3 Further Results on Definitional Extensions

In some cases the reverse implication of Lemma 1.60 is also true. We proceed towards this result by some preleminary observations.

defi:expansion

Definition 1.65 (Expansion). Let $\Sigma_0 \subseteq \Sigma_1$ be signatures, a Σ_1-structure $\mathscr{M}_1 = (M_1, I_1)$ is called an *expansion* of a Σ_0-structure $\mathscr{M}_0 = (M_0, I_0)$ if $M_0 = M_1$ and for all $f, p \in \Sigma_0$ $I_1(f) = I_0(f)$ and $I_1(p) = I_0(p)$.

lemma:coincidence

Lemma 1.66 (Coincidence Lemma). *Let $\Sigma_0 \subseteq \Sigma_1$ be signatures, and $\phi \in Fml_{\Sigma_0}$. Furthermore let $\mathscr{M}_0$ be a Σ_0-structure and $\mathscr{M}_1$ an Σ_1-expansion of $\mathscr{M}_0$. Then*

$$\mathscr{M}_0 \models \phi \quad \Leftrightarrow \quad \mathscr{M}_1 \models \phi$$

Proof

Obvious. □This lemma says that the truth or falisity of a sentence ϕ in a given structure only depends on the symbols actually occuring in ϕ. It is hard to imagine a logic where this would not hold true. There are in fact, rare cases, e.g., a typed first-order logic with a type hierachy containing subtypes and abstract types, where the coincidence lemma does not apply.

defi:substructure

Definition 1.67 (Substructure). Let $\mathscr{M} = (M, I)$, $\mathscr{M}_0 = (M_0, I_0)$ be Σ-structures. $\mathscr{M}_0$ is called a *substructure* of $\mathscr{M}$ iff

1. $M_o \subseteq M$
2. for every n-ary function symbol $f \in \Sigma$ and any n of elements $a_1, \dots, a_n \in M_0$

$$I(f)(a_1, \dots, a_n) = I_0(f)(a_1, \dots, a_n)$$

3. for every n-ary relation symbol $p \in \Sigma$ and any n of elements $a_1, \dots, a_n \in M_0$

$$(a_1, \dots, a_n) \in I(p) = (a_1, \dots, a_n) \in I_0(p)$$

lemma:substructure

Lemma 1.68. *Let $\mathscr{M}_0$ be a substructure of $\mathscr{M}$ and ϕ logically equivalent to a universal sentence. Then*

$$\mathscr{M} \models \phi \Rightarrow \mathscr{M}_0 \models \phi$$

Proof

Easy induction on the complexity of ϕ. □

defi:MSigma

Definition 1.69. Let $\mathscr{M}$ be a Σ-structure.
The signature Σ_M is obtained from Σ by adding new constant symbols c_a for every element $a \in M$.

The expansion of $\mathscr{M}$ to a Σ_M-structure $\mathscr{M}^* = (M, I^*)$ is effected by the obvious $I^*(c_a) = a$.

defi:diagram

Definition 1.70 (Diagram of a structure). Let $\mathscr{M}$ be a Σ-structure. The diagram of $\mathscr{M}$, in symbols $Diag(\mathscr{M})$, is defined by

$$Diag(\mathscr{M}) = \{\phi \in Fml_{\Sigma_M} \mid \mathscr{M}^* \models \phi \text{ and } \phi \text{ is quantierfree}\}$$

lemma:diag

Lemma 1.71. *Let $\mathscr{M}$ be a Σ-structure.*
If $\mathscr{N} \models Diag(\mathscr{M})$ then $\mathscr{M}$ is (isomorphic to) a substructure of $\mathscr{N}$.

Proof

Easy. □

lemma:CExt->SemCExt

Lemma 1.72. *Let $\Sigma_0 \subseteq \Sigma_1$ be signatures, and T_i sets of sentences in Fml_{Σ_i} and assume that*

1. *T_1 contains only universal sentences and*
2. *$\Sigma_1 \setminus \Sigma_0$ contains only relation symbols.*

If T_1 is a conservative extension of T_0
then T_1 is also a semantic conservative extension of T_0

Proof

We need to show the two clauses in Definition 1.59.
(1): Let $\mathscr{M}_1$ be a Σ_1-structure with $\mathscr{M}_1 \models T_1$ and $\mathscr{M}_0$ its restriction to Σ_0, i.e., $\mathscr{M}_0 = \mathscr{M}_1 \upharpoonright \Sigma_0$. For all $\phi \in T_0$ obviously $T_0 \vdash \phi$. Thus also $T_1 \vdash \phi$ and therefore $\mathscr{M}_1 \models \phi$. By the coincidence lemma this gives $\mathscr{M}_0 \models \phi$. Thus, we get $\mathscr{M}_0 \models T_0$ as desired.
(2): Here we look at a Σ_0-structure $\mathscr{M}_0$ with $\mathscr{M}_0 \models T_0$. We set out to find an expansion $\mathscr{M}_1$ of $\mathscr{M}_0$ with $\mathscr{M}_1 \models T_1$. To this end we consider the theory $T_1 \cup Diag(\mathscr{M}_0)$. If this theory were inconistent than already $T_1 \cup F$ for a finite subset $F \subseteq Diag(\mathscr{M}_0)$ would be inconsistent. This is the same as saying $T_1 \vdash \neg F$. Since the constants c_a do not occur in T_1 we get furthermore $T_1 \vdash \forall x_1,\ldots,x_n \neg F'$, where F' is obtained from F be replacing all occurences of constants c_a by the same variable x_i. This is equivalent to $T_1 \vdash \neg\exists x_1,\ldots,x_n F'$. Since T_1 was assume to be a conservative extension of T_0 we also get $T_0 \vdash \neg\exists x_1,\ldots,x_n F'$ and thus $\mathscr{M}_0 \models \neg\exists x_1,\ldots,x_n F'$. This is a contradiction since by the definition of $Diag(\mathscr{M}_0)$ we have $\mathscr{M}_0 \models \exists x_1,\ldots,x_n F'$ by instantiating the quantified variable x_i that replaces the constant c_a by the element a. This contradiction shows that $T_1 \cup Diag(\mathscr{M}_0)$ is consistent. Let $\mathscr{N}$ be a model of this theory. By Lemma 1.71 we may assume that $\mathscr{M}_0$ is a substructure of $(\mathscr{N} \upharpoonright \Sigma_0)$. Since by assumption only new relation symbols are added when passing from Σ_0 to Σ_1 also $(\mathscr{N} \upharpoonright \Sigma_1)$ is a substructure of $\mathscr{N}$. By Lemma 1.68 we get $(\mathscr{N} \upharpoonright \Sigma_1) \models T_1$. Obviously, $(\mathscr{N} \upharpoonright \Sigma_1)$ is an expansion of $(\mathscr{N} \upharpoonright \Sigma_0) = \mathscr{M}_0$ and we are finished. □

1.5.4 Relative Completeness of $CoT_{seq} + seqDepth$

To formulate the main result of this subsection we need the following auxiliary definition

defi:SeqFreeStructure

Definition 1.73. We call a structure $\mathscr{M} = (M, \delta, I)$ an *sequence free* structure if

1. it satisfies all the restrictions on JFOL structure except Definitions 1.52, 1.53, 1.54,

2. the axioms from Figures 1.15, 1.17, and 1.16 are true in $\mathscr{M}$.

In an sequence free structure $\mathscr{M}$ the domains for types other than *Seq* are still fixed, in particular the type domain for *Int* is $\mathbb{Z}$. The next theorem shows that the axioms are strong enough to enforce that the restrictions from Definitions 1.52, 1.53, 1.54 are satisfied. The proof depends, among others, on the fact that the type domain of *Int* is $\mathbb{Z}$.This is, why we call the result a *relative* completeness result.

Theorem 1.74 (Relative Completness).

thm:seqDepthTheoryComplete *Every sequence free structure $\mathscr{M}$ is isomorphic to a JFOL structure.*

Proof. Let $\mathscr{N}$ be an arbitrary sequence free structure. We will construct an isomorphism $F : \mathscr{N} \to \mathscr{M}$ for a JFOL structure $\mathscr{M}$. By definition of a sequence free structure the type universes in $\mathscr{N}$ coincide with the type universes in any JFOL structure for all types except *Seq*. So let F be the identity function on all those type universes. It remains to define F from the domain $Seq^{\mathscr{N}}$ for type *Seq* in $\mathscr{N}$ into the type domain D^{Seq} that is common to all JFOL structures. $F(a)$ is defined by induction on $seqDepth(a)$. If $seqDepth(a) = 0$ then the axiom from Figure 1.16 implies that $a \notin Seq^{\mathscr{N}}$ and we had already stipulated $F(a) = a$ in this case. For a with $seqDepth^{\mathscr{N}}(a) = n+1$ we define inductively

$$F(a) = \langle F(a[0]), \ldots, F(a[k-1]) \rangle$$

with $k = seqLen^{\mathscr{N}}(a)$, $a[i]$ shorthand for $any :: seqGet^{\mathscr{N}}(a,i)$. Since $\mathscr{N}$ satisfies the axiom from Definition in Figure 1.16 we know $seqDepth^{\mathscr{N}}(a[i]) \leq n$ and this definition is really a valid recursion.

From the core axiom e*qualityToSeqGetAndSeqLen* in Figure 1.15 we get immediately that F thus defined is an injective function. We want to argue that F is also surjective. We will exhibit for every $a \in D^n_{Seq}$, by induction on n, a term t such that $F(t^{\mathscr{N}}) = a$. For $n = 0$, we know that $a \notin Seq^{\mathscr{N}}$ and $F(a) = a$. In the inductive step of the argument we assume that the claim is true for all $a \in D^n_{Seq}$ and fix $s = \langle s_0, \ldots s_{k-1} \rangle \in D^{n+1}_{Seq}$. Since $s_i \in D^n_{Seq}$ for all $0 \leq i < k$ there are elements a_i with $F(a_i) = s_i$. We define a term t with fresh new variables $x_0, \ldots, x_{k-1}$:

$$\begin{aligned} t = {} & \textbf{if } u = 0 \textbf{ then } x_0 \textbf{ else} \\ & (\textbf{if } u = 1 \textbf{ then } x_1 \textbf{ else} \\ & \ldots \\ & (\textbf{if } u = k-1 \textbf{ then } x_{k-1}) \ldots) \end{aligned}$$

Inzerpreting variable x_i with a_i we obtain. $F((seqDef\{u\}(0,k,t))^{\mathscr{N}}) = s$. Note, that axiom g*etOfSeqDef* from Figure 1.15 allows for variables in t. In total he have verified

$$F : N \to M \text{ is a bijection} \qquad (1.29)$$ align:IsoBij

So far we have for the range of F only made use of properties that are true for all JFOL structures. Now, we fix a particular JFOL structure $\mathscr{M}$ by defining $seqGetOutside^{\mathscr{M}} = F(seqGetOutside^{\mathscr{N}})$.

We need to verify the isomorphism properties of F with respect to the functions and predicates declared for the data type *Seq*.

$$\begin{aligned}
seqGet^{\mathcal{M}}_{Any}(F(s),i) &= seqGet^{\mathcal{M}}_{Any}(\langle F(a_0),\dots,F(a_{k-1})\rangle,i) && \text{Def. of } F\\
&= F(a_i) \quad \text{if } 0 \leq i < k && \text{Def. of } F(seqGet^{\mathcal{M}}_{Any})\\
&= F(seqGet^{\mathcal{N}}_{Any}(s,i)) && \text{Def.of } a_i \text{ in Def.of } F\\
&= seqGetOutside^{\mathcal{M}} \quad \text{if } \neg(0 \leq i < k) && \text{Def. of } seqGet^{\mathcal{M}}_{Any}\\
&= F(seqGetOutside^{\mathcal{N}}) && \text{Def. of } \mathcal{M}
\end{aligned}$$

In total we have shown $F(seqGet^{\mathcal{N}}_{Any}(s,i)) = seqGet^{\mathcal{M}}_{Any}(F(s),F(i))$ remembering that $F(i) = i$ for integers i.

$$\begin{aligned}
seqLen^{\mathcal{M}}(F(s)) &= seqLen^{\mathcal{M}}(\langle F(a_0),\dots,F(a_{k-1})\rangle) && \text{Def. of } F\\
&= k && \text{Def. of } seqLen^{\mathcal{M}}\\
&= seqLen^{\mathcal{N}}(s) && \text{Def.of } k \text{ in Def.of } F
\end{aligned}$$

In total $F(seqLen^{\mathcal{N}}(s)) = seqLen^{\mathcal{M}}(F(s))$.

By induction on n we show that $seqDepth^{\mathcal{N}}(s) = n$ iff $seqDepth^{\mathcal{M}}(s) = n$. For $n = 0$ we have $seqDepth^{\mathcal{N}}(s) = 0$ iff $s \notin Seq^{\mathcal{N}}$ iff $F(a) = s \notin Seq^{\mathcal{M}}$ iff $seqDepth^{\mathcal{M}}(s) = 0$. If $seqDepth^{\mathcal{N}}(s) = n+1$ and $a_i = seqGet^{\mathcal{N}}_{Any}(s)$ for $0 \leq i < seqLen^{\mathcal{N}}$ then we know by the axiom in Figure 1.16

$$\begin{aligned}
&seqDepth^{\mathcal{N}}(a_i) \leq n \quad \text{for all } 1 \leq i < seqLen^{\mathcal{N}}\\
&seqDepth^{\mathcal{N}}(a_j) = n \quad \text{for one } 1 \leq j < seqLen^{\mathcal{N}}
\end{aligned}$$

By definition of F we have $F(s) = \langle F(a_0),\dots,f(a_{k-1})\rangle$ with $k = seqLen^{\mathcal{N}}(s)$. By induction hypothesis we obtain

$$\begin{aligned}
&seqDepth^{\mathcal{M}}(F(a_i)) \leq n \quad \text{for all } 1 \leq i < seqLen^{\mathcal{N}}\\
&seqDepth^{\mathcal{M}}(F(a_j)) = n \quad \text{for one } 1 \leq j < seqLen^{\mathcal{N}}
\end{aligned}$$

which again by the defining axiom for *seqDepth* now applied for the structure $\mathcal{M}$ yields $seqDepth^{\mathcal{M}}(s) = n+1$.

We omit the proof of the isomorphism property for the remaining 11 functions (see Figure 1.14). They are easy since their interpretations in both $\mathcal{N}$ and $\mathcal{M}$ are fixed by the same definitional extensions. In case of partial functions, we still have a choice in the precise definition of the JFOL structure $\mathcal{M}$, which we use to make the isomorphism property true.

Figure 1.18 lists some consequences that can be derived from the definitions in Figure 1.17 and the Core Theory. The entry 1 is a technical lemma that is useful in the derivation of the following lemmas in the list. The entry 2 clarifies the role of the default value *seqGetOutside*; it is the default or error value for any out-of-range access. Rules 2 to 7 are schematic rule. These rules are applicable for any instantiations of the schema variable α by a type. Entry 3 addresses an important issue: on

1 seqSelfDefinition
$\forall Seq\ s; (s \doteq seqDef\{u\}(0, seqLen(s), seqGet_{Any}(s,u)))$

2 seqOutsideValue
$\forall Seq\ s; (\forall int\ i; ((i < 0 \vee seqLen(s) \leq i) \rightarrow seqGet_{\alpha}(s,i) \doteq (\alpha)seqGetOutside))$

3 castedGetAny
$\forall Seq\ s; \forall int\ i; ((\beta)seqGet_{Any}(s,i) \doteq seqGet_{\beta}(s,i))$

4 getOfSeqSingleton
$\forall Any\ x; \forall int\ i; (seqGet_{\alpha}(seqSingleton(x),i) \doteq \text{if } i \doteq 0 \text{ then } (\alpha)x \text{ else } (\alpha)seqGetOutside)$

5 getOfSeqConcat
$$\begin{aligned}\forall Seq\ s,s2; \forall int\ i; (seqGet_{\alpha}(seqConcat(s,s2),i) \doteq\ & \text{if } i < seqLen(s)\\ & \text{then } seqGet_{\alpha}(s,i)\\ & \text{else } seqGet_{\alpha}(s2, i - seqLen(s)))\end{aligned}$$

6 getOfSeqSub
$$\begin{aligned}\forall Seq\ s; \forall int\ from,to,i; (seqGet_{\alpha}(seqSub(s,from,to),i) \doteq\ & \text{if } 0 \leq i \wedge i < (to - from)\\ & \text{then } seqGet_{\alpha}(s, i + from)\\ & \text{else } (\alpha)seqGetOutside)\end{aligned}$$

7 getOfSeqReverse
$\forall Seq\ s; \forall int\ from,to,i; (seqGet_{\alpha}(seqReverse(s),i) \doteq seqGet_{\alpha}(s, seqLen(s) - 1 - i))$

8 lenOfSeqEmpty
$seqLen(seqEmpty) \doteq 0$

9 lenOfSeqSingleton
$\forall Any\ x; (seqLen(seqSingleton(x)) \doteq 1)$

10 lenOfSeqConcat
$\forall Seq\ s,s2; (seqLen(seqConcat(s,s2)) \doteq seqLen(s) + seqLen(s2))$

11 lenOfSeqSub
$\forall Seq\ s; \forall in\ from,to; (seqLen(seqSub(s,from,to)) \doteq \text{if } from < to \text{ then } (to - from) \text{ else } 0)$

12 lenOfSeqReverse
$\forall Seq\ s; (seqLen(seqReverse(s)) \doteq seqLen(s))$

13 seqConcatWithSeqEmpty
$\forall Seq\ s; (seqConcat(s, seqEmpty) \doteq s)$

14 seqReverseOfSeqEmpty
$seqReverse(seqEmpty) \doteq seqEmpty$

Fig. 1.18 Some derived rules for finite sequences fig:DerivedSeqMath

one hand there is the family of function symbols $seqGet_{\alpha}$, on the other hand there are the cast expressions $(\alpha)seqGet_{Any}$. The lemma says that both coincide. The entries 4 to 12 allow to determine the access function and the length of the empty sequence, singleton, concatenation, subsequence and reverse constructors. The last two entries 13 and 14 are examples for a whole set of rules that cover corner cases of the constructors involved.

Figure 1.19 lists some derived rules for the one-place predicate *seqNPerm* and the two-place predicate *seqPerm* that follow from the definitions in Figure 1.17. Surprisingly, none of the proofs apart from the one for seqNPermRange needs induction. This is mainly due to the presence of the $seqDef\{\}(,,)$ construct. The lemma seqNPermRange itself is a kind of pigeon-hole principle and could only be proved via induction.

1 seqNPermRange
$\forall Seq\ s; (seqNPerm(s) \rightarrow$
$\forall int\ i; (0 \leq i \wedge i < seqLen(s) \rightarrow (0 \leq seqGet_{int}(s,i) \wedge seqGet_{int}(s,i) < seqLen(s))))$

2 seqNPermInjective
$\forall Seq\ s;\ (seqNPerm(s) \wedge$
$\quad \forall int\ i, j; (0 \leq i \wedge i < seqLen(s) \wedge 0 \leq j \wedge j < seqLen(s) \wedge seqGet_{int}(s,i) \doteq seqGet_{int}(s, j))$
$\qquad \rightarrow i \doteq j)$

3 seqNPermEmpty
$seqNPerm(seqEmpty)$

4 seqNPermSingleton
$\forall int\ i; (seqNPerm(seqSingleton(i)) \leftrightarrow i \doteq 0)$

5 seqNPermComp
$\forall Seq\ s1, s2; (seqNPerm(s1) \wedge seqNPerm(s2) \wedge seqLen(s1) \doteq seqLen(s2) \rightarrow$
$seqNPerm(seqDef\{u\}(0, seqLen(s1), seqGet_{int}(s1, seqGet_{int}(s2,u)))))$

6 seqPermTrans
$\forall Seq\ s1, s2, s3; (seqPerm(s1,s2) \wedge seqPerm(s2,s3) \rightarrow seqPerm(s1,s3))$

7 seqPermRefl
$\forall Seq\ s; (seqPerm(s,s))$

Fig. 1.19 Some derived rules for permutations fig:DerivedPermMath

Applications of the theory of finite sequences can be found in Section 16.5 and foremost in the chapter on Radix Sort in the KeY book.

References

EbbinghausFlumThomas07 Heinz-Dieter Ebbinghaus, Jörg Flum, and Wolfgang Thomas. *Einführung in die mathematische Logik (5. Aufl.).* Spektrum Akademischer Verlag, 2007. 64

Gallier87 Jean H. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Wiley, 1987. 6, 13

Giese05 Martin Giese. A calculus for type predicates and type coercion. In Bernhard Beckert, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX 2005, Koblenz, Germany. Proceedings*, volume 3702 of *LNCS*, pages 123–137. Springer, 2005. 13, 15

Greenleaf00 F.P. Greenleaf. Notes on algebra, section 2: the system of integers. 2000-2008. 40

KirbyParis82 Laurie Kirby and Jeff Paris. Accessible independence results for Peano Arithmetic. *Bulletin of the London Mathematical Society*, 14(4), 1982. 35

McCarthy62 John McCarthy. Towards a mathematical science of computation. In Cicely M. Popplewell, editor, *Information Processing 1962, Proceedings of IFIP Congress 62, Munich, Germany*, pages 21–28. North-Holland, 1962. 36

Monk76 J. Donald Monk. *Mathematical Logic*, volume 37 of *Graduate Texts in Mathematics*. Springer, 1976. 64

SchmittUlbrich15 Peter H. Schmitt and Mattias Ulbrich. Axiomatization of typed first-order logic. In Nikolaj Bjørner and Frank de Boer, editors, *FM 2015: Formal Methods - 20th International Symposium, Oslo, Norway. Proceedings*, volume 9109 of *LNCS*, pages 470–486. Springer, 2015. 54

Shoenfield67 Joseph R. Shoenfield. *Mathematical Logic*. Addison–Wesley Publ. Comp., Reading, Massachusetts, 1967. 64

Ulbrich13 Mattias Ulbrich. *Dynamic Logic for an Intermediate Language. Verification, Interaction and Refinement*. PhD thesis, Karlsruhe Institut für Technologie, KIT, 2013. 31