Preface

I take the central task of *theoretical computing science (*TCS*)* to be the construction of mathematical models of computational phenomena. Such models provide us with a deeper understanding of the nature of *computation* and *representation*. For example, the early work on computability theory provided a mathematical model of computation. Later work on the semantics of programming languages enabled a precise articulation of the underlying differences among programming languages and led to a clearer understanding of the distinction between semantic representation and implementation. Early work in complexity theory supplied us with abstract notions that formally articulated informal ideas about the resources used during computation. Such mathematical modeling provides the means of exploring the properties and limitations of languages and tools that would otherwise be unavailable.

The aim of this book is to contribute to this fundamental activity. Here we have two interrelated goals. One is to provide a logical framework and foundation for the process of specification and the design of specification languages. The second is to employ this framework to introduce and study *computable models*. These extend the notion of specification to the more general arena of mathematical modeling where our aim is to build mathematical models that are constructed from specifications.

During the preparation of this book, every proper computer scientist at the University of Essex provided valuable feedback. Some provided quite detailed comments. I will not single out any of you; you know who you are. But to all who contributed, thank you. Referees on the various journal papers that led to the book also provided valuable advice and criticism. But my greatest debt is to my wife, Rosana. Over the years, she has read draft after draft and made innumerable (not literally) suggestions for change and improvement. Without her, the size of the set of errors that remains would be much greater than it is.

Chapter 2
Typed Predicate Logic

In this initial technical chapter we develop the logical framework within which to articulate our theories of data types (**TDT)**. It is also to form our basic language of specification and provide the host for the construction of computable models. It is important to note that we are not advocating a single theory of types, but a broad framework in which a rich variety of theories can be easily and elegantly formulated.

Generally in logic and theoretical computer science, type theories are inductively generated from some basic types via type constructors. Our framework needs to be sufficiently flexible to elegantly support a wide range of such constructors, including dependent types, subtypes, and polymorphism. In addition, it must support a *type of types*; i.e., it must facilitate a natural formulation of theories where objects used to classify data become themselves items of data. However, the standard approach to the syntax of logical languages, where the syntax is given via some context-free grammar, does not easily support the expression of such a wide variety of notions. Nor does the traditional approach to simple type theory, i.e., where the types are hard-wired to the terms.

However, computer science with its emphasis on types [4, 1] and type checking, presents us with a more flexible way of formulating a typed syntax.[1] Indeed, modern logical systems also display such grammatical flexibility, e.g., the type theories of Martin Löf [3, 2]. The present logical framework follows suit. In the next few sections we shall present it and explore its simple properties.

2.1 Judgments and Contexts

We employ a system of natural deduction that we shall call *typed predicate logic* (**TPL**). This will form the logical skeleton of all our theories. However, unlike standard logical systems, where there is only one judgment form in conclusions, we admit several. More exactly, it is a many-sorted natural deduction system with the following four judgment forms:

[1] This was itself inspired by early work in combinatorial logic.

R. Turner, *Computable Models*, DOI 10.1007/978-1-84882-052-4_2,

$$T\ type$$
$$\phi\ prop$$
$$t : T$$
$$\phi$$

The first asserts that T is a *type*, the second that ϕ is a *proposition*, the third that t is an object term of type T, and the fourth that ϕ is true. We shall refer to the first three as type-inference judgments.

These judgments are formed from a syntax of terms that are built from variables (x_0, x_1, x_2, x_3...), constant, function, and relation symbols, including equality ($=$), and the logical connectives (Ω, $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\forall$, $\exists$). As metavariables for strings on these alphabets, we employ the Roman and Greek alphabets, where we reserve x, y, z, u, v, w to range over the object-level variables of the language. While this is the stuff of the syntax, the actual grammar is determined not by a traditional BNF or context-free syntax, but by a type-inference system that is constituted by the *membership* and *formation* rules for types and propositions [6, 3, 5]. The rules for this rule-based grammar will form part of the overall proof system.[2]

Generally, judgments in the logic are made relative to a *context* Γ that is a finite sequence of terms. In the logic, these take one of the following two forms:

$$x : T$$
$$\phi$$

i.e., a *declaration* that a variable has a given type or the assumption that a proposition, ϕ, is true. Thus, sequents in the theory have the shape,

$$\Gamma \vdash \Theta$$

where Θ is one of our four judgment forms and Γ a context. Such sequents are the basic carriers of meaning in the logic. They determine not only what follows from what, but also what is grammatically legitimate. We shall call contexts that contain only type assignments, i.e., ones of the form $x : T$, *declaration contexts*. We shall

---

[2] This background syntax may be further refined via the following BNF grammar.

$$t ::= F(t_1, . - ., t_n) | R(t_1, . - ., t_n) | O(t_1, . - ., t_n) \mid t =_t t$$
$$t \vee t | t \wedge t | \neg t | \forall x : t \cdot t | \exists x : t . t$$

Similarly, the raw syntax of contexts might also be made explicit as follows.

$$\Gamma ::= t \mid t, \Gamma$$

However, such BNF style definitions do not play too much of a role, since they sanction way too much nonsense. They only provide the background strings for the actual grammar, which is rule-given, i.e., by the rules of the logic itself.

use c, c', d, d', etc., as variables for these contexts and c_Γ for that part of the context Γ that consists of just its type assignments.

2.2 Structural Rules

We begin with the structural rules, i.e., *assumption*, *thinning*, and *substitution*. The first two permit the addition of new (grammatically acceptable) assumptions. The next allows weakening under the same grammatical constraints. The final rule is a substitution rule. Note that it respects the fact that, in contexts, the order of the occurrence of assumptions is significant.

$$\mathbf{A_1} \qquad \frac{\Gamma \vdash T\ type}{\Gamma, x : T \vdash x : T} \qquad\qquad \mathbf{A_2} \qquad \frac{\Gamma \vdash \phi\ prop}{\Gamma, \phi \vdash \phi}$$

$$\mathbf{W}_1 \qquad \frac{\Gamma, \Delta \vdash \Theta \qquad \Gamma \vdash T\ type}{\Gamma, x : T, \Delta \vdash \Theta}$$

$$\mathbf{W}_2 \qquad \frac{\Gamma, \Delta \vdash \Theta \qquad \Gamma \vdash \phi\ prop}{\Gamma, \phi, \Delta \vdash \Theta}$$

$$\mathbf{Sub} \qquad \frac{\Gamma, y : S, \Delta \vdash \Theta \qquad \Gamma \vdash s : S}{\Gamma, \Delta[s/y] \vdash \Theta[s/y]}$$

where in $\mathbf{A_1}$ and $\mathbf{W_1}$, x is fresh (i.e., it is not declared in Γ, Δ) and $\Theta[s/y]$ indicates the substitution of the term s for the variable y. Grammatical constraints play a significant role. For example, in $\mathbf{A_1}$ we are only permitted to add type assignments involving terms that are types, whereas $\mathbf{A_2}$ only sanctions assumptions that are propositions. Note that we do not have an exchange rule. This is a consequence of the fact that, in general, contexts will be dependent, i.e., the grammatical status of later assumptions may depend upon earlier ones. For example, the status of a purported proposition may depend upon the types of its free variables, and so may depend upon previous type declarations. A simple illustration of such dependence is generated by the equality rules. We shall see this shortly. The **Sub** rule could be avoided (it is partly covered by the universal elimination rule), but it will often prove convenient for the statement of our theories.

2.3 Types

As we have previously emphasized, types, relations, and functions are the basic building blocks of computable models. However, our treatment is not standard. They

are to be taken as intensional and primitive notions whose content is given by the rules of the system. This will become clear as we proceed.

The types of any particular theory will be given in terms of some basic types and closed under a collection of type constructors. More explicitly, in their most elementary guise, the formation rules for types will take the following shape.

$$\mathbf{O}_1 \quad B \; type \qquad\qquad \mathbf{O_2} \quad \frac{\Gamma \vdash T_1 \; type, ..., \Gamma \vdash T_n \; type}{\Gamma \vdash O(T_1, ..., T_n) \; type}$$

The first rule allows for the inclusion of basic types such as Booleans and numbers. In addition, there will be a type rule for each type constructor O of the language. For example, the rule for Cartesian products has the following formation rule.

$$\frac{\Gamma \vdash T \; type \qquad\qquad \Gamma \vdash S \; type}{\Gamma \vdash T \otimes S \; type}$$

Such a rule might be expressed in standard context-free style as follows.

$$type ::= ... | \; type \otimes type$$

But this approach is limited in that it does not easily support dependency. Later, we shall consider generalizations of such type formation rules as $\mathbf{O}_2$, rules that permit types to depend upon propositions and other types, i.e., dependent types.[3] We shall discuss these notions in more detail when we get to them. Our rule-based account of the grammar of types will really come into its own when we consider a type of types and, subsequently, consider the types themselves as items of data. Here we indicate their possibility to give the reader a sense of what is to come and that this is a much more flexible approach to type formation than any standard context-free style grammar. One cannot easily generalize the latter to cater for such notions of type.

The alert reader might think that we also need rules of type equality. We shall get to these when types are themselves taken to be objects in the theory. At this point, suffice it to say that, whatever type equality is taken to be, it will not be extensional; i.e., we shall not identify two types on the basis of shared membership.

---

[3] For example, in the following the first rule generalizes Cartesian products to allow for the second type to depend on the first one. The second introduces separation or sub-types. Here proposition formation may depend upon type formation, and subsequently, type formation may depend upon proposition formation.

$$\frac{\Gamma \vdash T \; type \qquad\qquad \Gamma, x : T \vdash S \; type}{\Gamma \vdash \Sigma x : T \cdot S \; type} \qquad\qquad \frac{\Gamma \vdash T \; type \qquad\qquad \Gamma, x : T \vdash \phi \; prop}{\Gamma \vdash \{x : T \cdot \phi\} \; type}$$

2.4 Relations and Functions

A relation is introduced by its grammatical rule, which takes the following shape.

$$\mathbf{R}\ \frac{\Gamma \vdash t_1 : T_1, ..., \Gamma \vdash t_n : T_n}{\Gamma \vdash R(t_1, ..., t_n)\ prop}$$

This informs us of its grammatical territory, its intended domain and range. Of course, in any theory there may be many such relations, given by such rules.

This notion of relation is to be seen in contrast to the standard set-theoretic one in which a relation is taken to be a set of ordered tuples i.e.,

$$R \subseteq T_1 \otimes, ..., \otimes T_n$$

where now $T_1, ..., T_n$ are sets. This is a fundamentally different notion of relation. And while ours has a set-theoretic interpretation, it is not the intended one. As we have said before, our relations are not taken to be extensional; i.e., they are not taken to satisfy any axiom of extensionality that insists that relations that hold of the same objects are the same relation. This is forced by the set-theoretic interpretation.

Similar remarks apply to functions. In the rule for function symbols, the resulting type is tied to a type constructor of the language; i.e., we assume that rule **O** governs the formation of the type $O(T_1, ..., T_n)$. Thus, the types themselves are introduced via rules and the function symbols follow suit.

$$\mathbf{F}\ \frac{\Gamma \vdash t_1 : T_1, ..., \Gamma \vdash t_n : T_n}{\Gamma \vdash F(t_1, ..., t_n) : O(T_1, ..., T_n)}$$

In line with the generalization of type formation, the rules for functions and relations will admit parallel generalizations. But we shall explain these in context.

2.5 Equality

The formation rule for equality is a special case of the formation rule for relations. $\mathbf{E_1}$ insists that equality forms a proposition when the terms flanking it have the same type. In addition, distinguished symbols such as equality are given content by their associated axioms and rules. $\mathbf{E_2}$ and $\mathbf{E_3}$ are the standard rules of introduction and elimination; i.e., every element of every type is equal to itself and equal objects can be substituted for each other in all contexts.

$$\mathbf{E_1}\ \frac{\Gamma \vdash t : T \qquad \Gamma \vdash s : T}{\Gamma \vdash t =_T s\ prop} \qquad \mathbf{E_2}\ \frac{\Gamma \vdash t : T}{\Gamma \vdash t =_T t}$$

$$\mathbf{E_3}\ \frac{\Gamma \vdash t =_T s \qquad \Gamma \vdash \Theta[t/x]}{\Gamma \vdash \Theta[s/x]}$$

The equality rules illustrate how dependency in contexts can occur. For instance, a provable sequent such as

$$x : T, y : T, z : T, x =_T y, y =_T z \vdash x =_T z$$

demonstrates how the occurrences of equality in the context (as well as the conclusion) are legitimate (i.e., form propositions) only where their constituent terms have the same type. Observe that, as a suffix, the type of the equality symbol is explicitly marked. However, where the context determines matters, we shall often drop the subscript on the equality relation; i.e., we shall just write:

$$x : T, y : T, z : T, x = y, y = z \vdash x = z$$

This principle of parsimony will be adopted generally.

2.6 Propositional Rules

We next provide the rules for the propositional connectives. The formation rules for these connectives capture their standard closure conditions, i.e., the ones normally given in a context-free style, while the introduction and elimination rules are their standard introduction and elimination logical rules.

$$\mathbf{L_1}\ \frac{\Gamma \vdash \phi\ prop \qquad \Gamma \vdash \psi\ prop}{\Gamma \vdash \phi \wedge \psi\ prop} \qquad \mathbf{L_2}\ \frac{\Gamma \vdash \phi \qquad \Gamma \vdash \psi}{\Gamma \vdash \phi \wedge \psi}$$

$$\mathbf{L_3}\ \frac{\Gamma \vdash \phi \wedge \psi}{\Gamma \vdash \phi} \qquad \mathbf{L_4}\ \frac{\Gamma \vdash \phi \wedge \psi}{\Gamma \vdash \psi}$$

$$\mathbf{L_5}\ \frac{\Gamma \vdash \phi\ prop \qquad \Gamma \vdash \psi\ prop}{\Gamma \vdash \phi \vee \psi\ prop}$$

$$\mathbf{L_6}\ \frac{\Gamma \vdash \phi \vee \psi \qquad \Gamma, \phi \vdash \eta \qquad \Gamma, \psi \vdash \eta}{\Gamma \vdash \eta}$$

$$\mathbf{L_7}\ \frac{\Gamma \vdash \phi \qquad \Gamma \vdash \psi\ prop}{\Gamma \vdash \phi \vee \psi} \qquad \mathbf{L_8}\ \frac{\Gamma \vdash \psi \qquad \Gamma \vdash \phi\ prop}{\Gamma \vdash \phi \vee \psi}$$

$$\mathbf{L_9}\quad \Gamma \vdash \Omega\ prop \qquad \mathbf{L_{10}}\quad \frac{\Gamma \vdash \phi \quad \Gamma \vdash \neg\phi}{\Gamma \vdash \Omega}$$

$$\mathbf{L_{11}}\quad \frac{\Gamma \vdash \phi\ prop \qquad \Gamma \vdash \Omega}{\Gamma \vdash \phi}$$

$$\mathbf{L_{12}}\quad \frac{\Gamma \vdash \phi\ prop \qquad \Gamma \vdash \psi\ prop}{\Gamma \vdash \phi \to \psi\ prop} \qquad \mathbf{L_{13}}\quad \frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \to \psi}$$

$$\mathbf{L_{14}}\quad \frac{\Gamma \vdash \phi \to \psi \quad \Gamma \vdash \phi}{\Gamma \vdash \psi}$$

$$\mathbf{L_{15}}\ \frac{\Gamma \vdash \phi\ prop}{\Gamma \vdash \neg\phi\ prop} \qquad \mathbf{L_{16}}\ \frac{\Gamma, \phi \vdash \Omega}{\Gamma \vdash \neg\phi} \qquad \mathbf{L_{17}}\ \frac{\Gamma, \neg\phi \vdash \Omega}{\Gamma \vdash \phi}$$

There are additional grammatical assumptions in some of the rules. For instance, in the disjunction introduction rules ($\mathbf{L_7}$, $\mathbf{L_8}$), we include the assumption that the alternate constituent of the disjunction has to be a proposition. These grammatical side conditions, as we shall see, are to ensure that only grammatically legitimate objects (i.e., propositions) are provable. Note that the underlying logic is classical logic. Unless overridden by parentheses, we shall assume that negation takes precedence over conjunction and disjunction, which take precedence over implication. But most of the time we shall use brackets.

2.7 Quantifier Rules

Aside from their generalized grammatical setting, the rules for the quantifiers are also classical. In particular, we assume the normal side conditions for the quantifier rules; i.e., in $\mathbf{L_{20}}$, x must not be free in Γ, T, or η, and in $\mathbf{L_{22}}$, x must not be free in any proposition in Γ.

$$\mathbf{L_{18}}\ \frac{\Gamma, x : T \vdash \phi\ prop}{\Gamma \vdash \exists x : T \cdot \phi\ prop}$$

$$\mathbf{L_{19}}\ \frac{\Gamma \vdash \phi[t/x] \quad \Gamma \vdash t : T \qquad \Gamma, x : T \vdash \phi\ prop}{\Gamma \vdash \exists x : T \cdot \phi}$$

$$\mathbf{L_{20}}\ \frac{\Gamma \vdash \exists x : T \cdot \phi \qquad \Gamma, x : T, \phi \vdash \eta}{\Gamma \vdash \eta} \qquad \mathbf{L_{21}}\ \frac{\Gamma, x : T \vdash \phi\ prop}{\Gamma \vdash \forall x : T \cdot \phi\ prop}$$

$$\mathbf{L_{22}}\ \frac{\Gamma, x : T \vdash \phi}{\Gamma \vdash \forall x : T \cdot \phi} \qquad \mathbf{L_{23}}\ \frac{\Gamma \vdash \forall x : T \cdot \phi \quad \Gamma \vdash t : T}{\Gamma \vdash \phi[t/x]}$$

We shall assume that the scope of the quantifier in $\forall x : T \cdot \phi$, $\exists x : T \cdot \phi$ is the whole of ϕ. It is only overridden by explicit parentheses.

This concludes the rules of **TPL**. We shall often indicate matters explicitly and write

$$\Gamma \vdash_{\mathbf{TPL}} \Theta$$

if the sequent $\Gamma \vdash \Theta$ is derivable using the rules of **TPL**.

The system may appear to be somewhat nonstandard, especially for the reader accustomed to first-order predicate logic. Hence, we provide some example derivations.

2.8 TPL Derivations

There is little here that is not a straightforward generalization that flows from the additional rules that replace the standard context-free grammar of a typed logic. However, given the slightly novel nature of **TPL**, we illustrate its notion of deduction with some simple examples. Of course, we shall see many more throughout the book. However, they will be somewhat less completely and formally presented.

Example 1 We deduce

$$\forall x : B \cdot \forall y : B \cdot x =_B y \to y =_B x$$

By the first equality rule, $\mathbf{E}_1$, we have

$$\frac{x : B, y : B \vdash x : B \qquad x : B, y : B \vdash y : B}{x : B, y : B \vdash x =_B y \ prop} \tag{1}$$

In the following, (2) is an instance of the structural rule, $\mathbf{A_2}$.

$$\frac{x : B, y : B \vdash x =_B y \ prop}{x : B, y : B, x =_B y \vdash x =_B y} \tag{2}$$

Step (3) is an instance of the second equality rule $\mathbf{E}_2$.

$$\frac{x : B \vdash x : B}{x : B \vdash x =_B x} \tag{3}$$

The conclusion of (3) may be enriched to (4). This follows by a judicious use of $\mathbf{A}_1$ and $\mathbf{A_2}$.

$$\frac{x : B \vdash x =_B x}{x : B, y : B, x =_B y \vdash x =_B x} \tag{4}$$

By the third equality rule, the conclusions of (2) and (4), we may deduce the following

$$\frac{x : B, y : B, x =_B y \vdash x =_B y \quad x : B \vdash x =_B x}{x : B, y : B, x =_B y \vdash y =_B x} \tag{5}$$

By the implication introduction rule $\mathbf{L}_{13}$ and the conclusion of (5), we can deduce (6).

$$\frac{x : B, y : B, x =_B y \vdash y =_B x}{x : B, y : B \vdash x =_B y \to y =_B x} \tag{6}$$

By $\mathbf{L}_{21}$ and the conclusion of (6), we may conclude

$$\frac{x : B, y : B \vdash x =_B y \to y =_B x}{x : B \vdash \forall y : B \cdot x =_B y \to y =_B x} \tag{7}$$

By the conclusion of (7) and $\mathbf{L}_{21}$, we arrive at the following

$$\frac{x : B \vdash \forall y : B \cdot x =_B y \to y =_B x}{\forall x : B \cdot \forall y : B \cdot x =_B y \to y =_B x} \tag{8}$$

Example 2 We deduce

$$\forall x : B \cdot \exists y : B \cdot x =_B y$$

By the first structural rule, $\mathbf{A}_1$, we have

$$\frac{B\ type}{x : B \vdash x : B} \tag{1}$$

By the first equality rule, $\mathbf{E}_1$, we have

$$\frac{x : B, y : B \vdash x : B \qquad x : B, y : B \vdash y : B}{x : B, y : B \vdash x =_B y\ prop} \tag{2}$$

The conclusion (3) is an instance of the equality rule $\mathbf{E_2}$

$$\frac{x : B \vdash x : B}{x : B \vdash x =_B x} \tag{3}$$

By the existential introduction rule, $\mathbf{L}_{19}$, and conclusions of (1), (2), and (3), we may deduce the following

$$\frac{x : B \vdash x =_B x \qquad x : B \vdash x : B \quad x : B, y : B \vdash x =_B y\ prop}{x : B \vdash \exists y : B \cdot x =_B y} \tag{4}$$

By the conclusion of (4) and universal introduction, we obtain

$$\frac{x : B \vdash \exists y : B \cdot x =_B y}{\forall x : B \cdot \exists y : B \cdot x =_B y} \tag{5}$$

We shall not see a great many examples worked out in such great detail. But these should be enough for the reader to grasp the dynamics of deduction in the system.

Example 3 Given

$$\phi \; prop \; and \; \psi \; prop$$

i.e., we can derive these in some context, we may define, in that context,

$$\phi \leftrightarrow \psi \triangleq (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$$

These is a new defined connective that illustrates the way that new notions are introduced via specification. But more of this later.

2.9 Type Inference

A distinctive aspect of **TPL** is its underlying type-inference system. As mentioned at the outset, we have inherited our approach to typed systems from the type-checking approach to syntax developed by computer scientists to ensure the type correctness of programs. It is a flexible approach in which types are not attached to terms. Instead, terms receive their types via type declarations.

This type-inference system constitutes the real grammar of **TPL**. We shall refer to it as **TI**. It is populated by the formation and type membership rules for the theory. Such a grammatical framework not only supports a very elegant and syntactically sensitive way of expressing a wide range of theories of data, but also has some conceptual significance. Types in our theories are meant to be vehicles for carving up the world in ways that can assist the computational model builder. As such, they play somewhat the same role as dimensional analysis in physics, a role that is isolated in the following subsystem.

Definition 4 *The subtheory* **TI** *is that sub theory of* **TPL** *whose rules are those of* **TPL** *but restricted to instances of the form*

$$\frac{c_1 \vdash \Theta_1, \ldots, c_n \vdash \Theta_n}{c \vdash \Theta}$$

where the contexts are type declarations and the conclusions (Θ, Θ_i) *are type-inference judgments, i.e., of the form*

$$T\ type$$
$$\phi\ prop$$
$$t : T$$

We write

$$c \vdash_{\mathbf{TI}} \Theta$$

if the sequent follows in **TI**.

A quick glance shows that only the rules O_1, O_2, R, F, E_1, E_3, A_1, W_1, W_2, Sub, L_1, L_5, L_9, L_{12}, L_{15}, L_{18}, and L_{21} furnish possible instances of such type-inference rules.

We first establish that this system is independent of the main one; i.e., it is a genuine subsystem.

Proposition 5 ***(Independence)*** *If* $\Gamma \vdash_{\mathbf{TPL}} \Theta$*, where* Θ *is a type-inference judgment, then* $c_\Gamma \vdash_{\mathbf{TI}} \Theta$.

Proof By induction on the rules with type-inference conclusions. Observation of these demonstrates that they only require declaration contexts and type-inference premises. For example, consider the structural rule

$$\frac{\Gamma \vdash T\ type}{\Gamma, x : T \vdash x : T}$$

By induction, $c_\Gamma \vdash T\ type$. By the rule itself, $c_\Gamma, x : T \vdash T\ type$. Similarly, for the following rule, i.e., if only type-inference is used in the premises, it is only used in the conclusion.

$$\frac{\Gamma, \Delta \vdash \Theta \qquad \Gamma \vdash T\ type}{\Gamma, x : T, \Delta \vdash \Theta}$$

This style of argument succeeds for all cases.■

The following provides the basis for a type-checking algorithm.

Proposition 6 ***(Type Checking)*** *In* ***TI*** *we have:*

1. $c \vdash R(t_1, ..., t_n)\ prop$ iff $c \vdash t_1 : T_1$ and...and $c \vdash t_n : T_n$,
2. $c \vdash O(T_1, ..., T_n)\ type$ iff $c \vdash T_1\ type$ and...and $c \vdash T_n\ type$,
3. $c \vdash F(t_1, ..., t_n) : O(T_1, ..., T_n)$ iff $c \vdash t_1 : T_1$ and...and $c \vdash t_n : T_n$,
4. $c \vdash \phi \circ \psi\ prop$ iff $c \vdash \phi\ prop\ and\ c \vdash \psi\ prop$, where $\circ\ = \vee, \wedge, \rightarrow$,
5. $c \vdash \neg\phi\ prop$ iff $c \vdash \phi\ prop$,
6. $c \vdash \mathcal{Q}x : T.\phi\ prop$ iff $c, x : T \vdash \phi\ prop$ where $Q = \exists$ or $\forall$
7. $c \vdash t =_T s\ prop$ iff $c \vdash t : T$ and $c \vdash s : T$

Proof The directions from right to left follow immediately from the rules. For the other direction, we use induction on the structure of derivations. Consider part 1. If the conclusion follows from the formation rule

$$\mathbf{R}\ \frac{c \vdash t_1 : T_1, ..., c \vdash t_n : T_n}{c \vdash R(t_1, ..., t_n)\ prop}$$

the result is immediate. If the conclusion is the result of a structural rule, the result follows from using the structural rule itself. Consider part 4. If the conclusion follows from the introduction rules for the connective, the result is immediate. If the conclusion is the result of a structural rule, the result follows from using the structural rule itself. For example, suppose the last step in the derivation is the following instance of an application of W_1.

$$\frac{c \vdash T\ type \quad c \vdash \phi \wedge \psi\ prop}{c, x : T \vdash \phi \wedge \psi\ prop}$$

Consider the premises. By induction, we may suppose that $c \vdash \phi\ prop$ and $c \vdash \psi\ prop$. By induction, $c, x : T \vdash \phi\ prop$ and $c, x : T \vdash \psi\ prop$.

The other rules follow exactly the same pattern of argument.■

Using the left-to-right directions, we obtain an obvious recursive algorithm for type checking. The next result is significant for the coherence of the logic. It guarantees that what is provable is grammatical.

Theorem 7 ***(Coherence)***

1. If $\Gamma \vdash \phi$, then $\Gamma \vdash \phi\ prop$,
2. If $\Gamma \vdash t : T$, then $\Gamma \vdash T\ type$,
3. If $\Gamma, x : T, \Gamma' \vdash \Theta$, then $\Gamma \vdash T\ type$,
4. If $\Gamma, \phi, \Gamma' \vdash \Theta$, then $\Gamma \vdash \phi\ prop$

Proof By induction on the structure of derivations. Most of the cases are routine. We illustrate part 1 with the cases of disjunction elimination, universal introduction, and existential quantification introduction. Consider

$$\frac{\Gamma \vdash \phi \vee \theta \qquad \Gamma, \phi \vdash \eta \qquad \Gamma, \theta \vdash \eta}{\Gamma \vdash \eta}$$

By induction, and using type checking, we obtain

$$c_\Gamma \vdash \eta\ prop$$

Next, consider the existential introduction rule

$$\frac{\Gamma \vdash \phi[t/x] \qquad \Gamma \vdash t : T \qquad \Gamma, x : T \vdash \phi\ prop}{\Gamma \vdash \exists x : T \cdot \phi}$$

By induction, $c_\Gamma, x : T \vdash \phi\ prop$. By the formation rule for the existential quantifier, we are finished. Finally, consider the existential elimination rule in the following case.

$$\frac{\Gamma \vdash \exists x : T \cdot \phi \qquad \Gamma, x : T, \phi \vdash \eta}{\Gamma \vdash \eta}$$

By the premises $\Gamma, x : T, \phi \vdash \eta$. By induction and type checking, and the fact that x is not free in η, $c_\Gamma \vdash \eta\ prop$. For part 2, we illustrate with rule **F;** i.e., suppose that the last step is

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash t_n : T_n}{\Gamma \vdash F(t_1, ..., t_n) : O(T_1, ..., T_n)}$$

By induction and the assumptions, $T_1\ type$, $T_2\ type$,...,$T_n\ type$. Hence by the rule **O,** $O(T_1, ..., T_n)\ type$. For part 3, the substantial case is

$$\mathbf{W}_1\ \frac{\Gamma, \Gamma' \vdash \Theta \qquad \Gamma \vdash T\ type}{\Gamma, x : T, \Gamma' \vdash \Theta}$$

It follows immediately that $\Gamma \vdash T\ type$. The same argument works for part 4 and

$$\mathbf{W}_2\ \frac{\Gamma, \Gamma' \vdash \delta \qquad \Gamma \vdash \phi\ prop}{\Gamma, \phi, \Gamma' \vdash \delta}$$

The rest of the rules can be established using similar observations.■

TPL is a generalization of a standard many-sorted logic in two ways. First, the types may be inductively generated, and so it generalizes the simple fixed structure of standard many-sorted logic. Second, and more importantly, the variables of the theory range freely over the types. This has the knock-on effect that the grammatical legitimacy of the various syntactic constructs not only depends upon the types, but also depends dynamically on them; i.e., the expressions are only well formed relative to an assignment of types to the variables. This is a *Curry* (after Haskell Curry) approach to typing [1]. And for the natural and elegant development of our quite rich range of theories, we need all this flexibility. So the slightly complex nature of our logical framework will eventually reap its rewards.

References

1. Barendregt, H.P. Lambda Calculus With Types. In: S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, (Eds), Handbook of Logic in Computer Science. Oxford Science Publications. Abramsky, S., Gabbay, D.M. and Maibaum, T.S.E., pp. 118–310, Oxford University Press, Oxford, 1992.
2. Beeson, M.J. Foundations of Constructive Mathematics, Springer-Verlag Berlin, 1985.

3. Martin-Lof, P. An intuitionistic theory of sets, predicative part. In Logic Colloquim, 73. North-Holland, Amsterdam, 1975.
4. Pierce, B.C. Types and Programming Languages. MIT Press, Cambridge, MA, 2002.
5. Thompson, S. Type Theory and Functional Programming. Addison-Wesley, Reading, MA, 1991.
6. Turner, R. Type inference for set theory. Theor. Comput. Sci. 266(1–2): 951–974, 2001.