THE LOGIC OF SPECIFICATION

Computable Models
Outline

I. Specifications
II. Typed Predicate Logic
III. Specification in TPL
IV. Theories off Types
V. Polymorphic Specifications
VI. The Type of Specifications
VII. Recursive Specifications
VIII. Correctness
IX. Computable Specifications
X. Axiomatic Definitions
I. SPECIFICATIONS
Stipulative Definitions

- They involve no commitment that the assigned meaning agrees with prior use –if any. One may stipulatively define terms as one sees fit.
- They are not correct or incorrect.
- Philosophical definitions are not stipulative: a lack of fit with existing usage is an objection.
Definitions in Computer Science

- There is a very special use of definitions in computer science i.e., the *specification* and construction of computational artefacts.
- Here stipulative definitions come into their own.
Different Expressions of Specification

- Informal Mathematics
- Formal Logic and Arithmetic
- Programming Languages
1. $G(x) \equiv 3x^3 + 2x^2 + 7x + 9$

2. $P[x, y] \equiv \exists z. y = x + z^2$

3. $F(0) \equiv 12$
   $F(n+1) \equiv 3F(n) + 8$
Birthday book

known: NAME

birthday: NAME → DATE

Known : dom birthday
Π - Calculus

\[ \text{CAR}(\text{talk}, \text{switch}) \stackrel{\text{uc1}}{=} \text{talk}.\text{CAR}(\text{talk}, \text{switch}) + \text{switch}(\text{talk}', \text{switch}').\text{CAR}(\text{talk}', \text{switch}') \]
Architectural Patterns

Class diagrams typically describe the different entities of a system as classes and the relation between these. This may for example include system parts and their relation system data interfaces of communicating parts messages and operations of interfaces.
Programs as Specifications

```plaintext
x := 0; y := 1;
while x < n do (x := x + 1; y := x * y)

Fac(0) = 1
Fac(n+1) = n+1 * Fac(n)
```
Software Architecture: ADL Perspective

- Software Architecture is a set of components and the connections among them.
  - components
  - connectors
  - configurations
  - constraints
Relations, Functions and Objects

- We take the view that stipulative definitions introduce new:
  1. Relations
  2. Functions
  3. Objects

- We shall now develop a more precise logical account.
II. TYPED PREDICATE LOGIC
Types in Computer Science

- Different notions of type employed
- Type-checking is employed.
- Rich development of theories of types.
- To cope with these differences, we shall use a flexible/dependent version of typed predicate logic (TPL).
TPL

- Typed relation and function symbols
  \[ x_1:T_1,\ldots,x_n:T_n \vdash R(x_1,\ldots,x_n) \text{ prop} \]
  \[ x_1:T_1,\ldots,x_n:T_n \vdash F(x_1,\ldots,x_n) : T \]

- The type \( \text{prop} \) closed under the standard connectives and typed Quantifiers
  \[ x:T \vdash \varphi \text{ prop} \]
  \[ x:T \vdash \varphi \text{ prop} \]

  \[ \forall x:T. \varphi \text{ prop} \]
  \[ \exists x:T. \varphi \text{ prop} \]
Inference Rules

\[ \begin{align*}
\varphi & \quad \psi \\
\hline
\varphi \land \psi & \\
\varphi \models \psi & \quad \varphi \implies \psi
\end{align*} \]

Coherence: If \( \Gamma \models \psi \) then \( \Gamma \models \psi \text{ prop} \)
Example Types

- Basic types: Numbers, ....
- Products, Finite sets, Polymorphic types, Abstract types, classes,.....

\[
\begin{align*}
T \text{ type} & \quad S \text{ type} \\
\hline
T \times S & \quad \text{Set}(T) \text{ type} \\
S \Rightarrow T & \quad T \text{ type}
\end{align*}
\]
Σ Propositions

- Atomic propositions – maybe some subclass of them.
- Closed under disjunction, conjunction, existential quantification.
- And special operators for specific types, e.g. bounded quantifiers for numbers \((\forall x < n. \varphi)\) and \((\forall x \in s. \varphi)\) finite sets.
III. SPECIFICATIONS IN TPL
Definitions

- Within TPL, specifications, expressed as definitions take the form

  \[ R \iff [x_1:T_1, \ldots, x_n:T_n . \phi[x_1, \ldots, x_n]] \]

- This is governed by

  \[ \forall x_i:T_1, \ldots, x_n:T_n . R(x_i, \ldots, x_n) \iff \phi[x_1, \ldots, x_n] \]

- The addition of such a relation results in new theory $TPL_R$. 
Functions and Definite Descriptions

1. If $\forall x:T. \exists! y:S. \phi[x,y]$
   Then we may introduce a new function symbol
   $F(x) = y \iff \phi[x,y]$

2. If $\exists! x:T. \beta[x]$
   Then we may introduce a object symbol
   $\mu x. \beta[x]$

Such that $\beta[\mu x. \beta[x]] \land \forall x:T. \beta[x] \Rightarrow (x = \mu x. \beta[x])$
Admissible Specifications

In the general theory of stipulative definitions, there are coherence/consistency constraints.

*Conservative*: By defining new things, one cannot deduce anything new about old ones.

*Eliminability*: Anything said about new things can be reduced to something said about old ones.
Constraints in TPL

Conservative: Any prop $\varphi$ of $\text{TPL}$
$\text{TPL}_R \models \varphi$ implies in $\text{TPL} \models \varphi$

Elimination: For any prop $\varphi$ of $\text{TPL}_R$ there is a prop $\psi$ of $\text{TPL}$ such that $\text{TPL}_R \models \varphi \leftrightarrow \psi$
IV. THEORIES OF TYPES
Ontology via Types

- The types reflect the underlying ontology of the theory.
- A richer ontology of types yields richer expressive power.
- The notion of type can be determined by the application.
Typed Set Theory

\[
\begin{align*}
S_0 & \quad \frac{T \text{ type}}{\text{Set}(T) \text{ type}} \\
S_1 & \quad \frac{T \text{ type}}{\emptyset_T : \text{Set}(T)} \\
S_2 & \quad \frac{a : T \quad b : \text{Set}(T)}{a \oplus T b : \text{Set}(T)} \\
S_3 & \quad \frac{\phi[\emptyset] \quad \forall x : T \cdot \forall y : \text{Set}(T) \cdot \phi[y] \rightarrow \phi[x \oplus T y]}{\forall x : \text{Set}(T) \cdot \phi[x]} \\
S_4 & \quad \frac{a : T \quad b : \text{Set}(T)}{a \oplus (a \oplus b) = a \oplus b} \\
S_5 & \quad \frac{a : T \quad b : T \quad c : \text{Set}(T)}{a \oplus (b \oplus c) = b \oplus (a \oplus c)} \\
S_6 & \quad \frac{a : T \quad b : \text{Set}(T)}{a \in_T b \text{ prop}} \\
S_7 & \quad \frac{a : T \quad b : \text{Set}(T)}{a \in a \oplus b} \\
S_8 & \quad \frac{a \in c \quad b : T \quad c : \text{Set}(T)}{a \in b \oplus c} \\
S_9 & \quad \frac{a \in \emptyset_T}{\Omega} \\
S_{10} & \quad \frac{a \in b \oplus c}{a = b \lor a \in c}
\end{align*}
\]
Topological Space in Typed Set Theory

**Topological Space**

\[ F : \text{Set}(\text{Set}(U)) \]

- \( \emptyset \in F \)
- \( U \in F \)
- \( G \subseteq F \Rightarrow \bigcup G \in F \)
- \( f, g \in F \Rightarrow f \cap g \in F \)
### Constructive Type Theory

<table>
<thead>
<tr>
<th>$T$ type</th>
<th>$S$ type</th>
<th>$x:T$</th>
<th>$y:T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T \ominus S$ type</td>
<td>$I[T,x,y]$ type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x:T \vdash S$ type</td>
<td>$x:T \vdash S$ type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\Sigma x:T.S$ type</td>
<td>$\Pi x:T.S$ type</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\[ p: \text{Process} \quad q: \text{Process} \]
\[ \text{-----------------------------} \]
\[ p \parallel q: \text{Process} \]
\[ p: \text{Process} \quad q: \text{Process} \]
\[ \text{-----------------------------} \]
\[ p \parallel q \equiv q \parallel p \]
V. POLYMORPHIC SPECIFICATIONS
Types of Polymorphism

- In specification and programming Languages all the different forms of polymorphism can be found.
- *Implicit*: The MIRANDA programming language
- *Explicit*: Second Order Lambda Calculus
- *Subtype*: Object oriented
Implicit Polymorphism

- **Implicit**: The MIRANDA programming language
- Here programs have no type information attached to them.
- Types are computed via type-inference engines that try and find the most general type.
- These are built into TPL.
Subtype Polymorphism

- Is a form of type polymorphism in which program constructs, typically subroutines or functions, written to operate on elements of the super type can also operate on elements of the subtype.
- If $S$ is a subtype of $T$, the sub typing relation is often written $S <: T$, to mean that any term of type $S$ can be safely used in a context where a term of type $T$ is expected.
- Occurs in Object-oriented languages
Explicit Polymorphism

- Second Order Lambda Calculus
- Abstraction over types.
- $\Pi X.T$
- $\lambda X.t$
Explicit Polymorphism in TPL

Add a universal type $U$

$T$ type

$T:U$

$T$ type
Polymorphic Set Union

Set Union

\[ u : U, \ x : \text{Set}(u), \ y : \text{Set}(u), \ z : \text{Set}(u) \]

\[ \forall w : u. \ w \in z \iff w \in x \lor w \in y \]
VI. THE TYPE OF SPECIFICATIONS
Definitions as First Class objects

- So far Definitions are part of the meta theory
- One may include them as a new type
The Theory SC(Σ)

\[ \frac{T \text{ type}}{S(T) \text{ type}} \]

\[ \frac{x : T \vdash \phi \text{ prop}}{[x : T | \phi] : S(T)} \]

\[ \frac{s : S(T) \quad t : T}{s(t) \text{ prop}} \]

\[ \frac{x : T \vdash \phi \text{ prop} \quad t : T}{[x : T | \phi](t)} \]

\[ \frac{x : T \vdash \phi \text{ prop} \quad t : T}{\phi[t/x]} \]
Higher-order Specifications

\[
f? : S(U), \quad g? : S(U), \quad h! : S(U)
\]

\[
h = [x : T \mid fx \land gx]
\]
VII. RECURSIVE SPECIFICATIONS
The Form of Recursive Definitions

- In the theory SC, assume that

\[ x : T, f : S(T) \vdash \phi[f, x] \text{ prop} \quad \text{R}_0 \]

where, when \( f \) occurs in \( \phi \), it occurs as a predicate.

- A recursive schema specification has the following form:

\[ R \triangleq [x : T | \phi[R, x]] \quad \text{Rec} \]

where \( \phi[R, x] \) is obtained by replacing every occurrence of \( f \) by \( R \).

- This is taken to introduce a new relation symbol that satisfies the following versions of \( \text{R}_1 \), \( \text{R}_2 \), and \( \text{R}_3 \).

\[ x : T \vdash R(x) \text{ prop} \quad \text{R}_1 \]

\[ \forall x : T \cdot \phi[R, x] \rightarrow R(x) \quad \text{R}_2 \]

\[ \begin{align*}
  x : T & \vdash \theta[x] \text{ prop} \\
  \forall x : T \cdot \phi[\theta, x] & \rightarrow \theta[x]
\end{align*} \quad \text{R}_3 \]

\[ \forall x : T \cdot R(x) \rightarrow \theta[x] \]
Recursion operator taken from Gödel's Functionals of finite type

\[ R \]

\[ u : \text{type}, \ x : u, \ f : S(N \otimes u \otimes u), \ y : N, z : u \]

\[ y = 0 \land z = x \]
\[ \lor \]

\[ y \neq 0 \land \exists w : u \cdot R(u, x, f, \text{pred}(y), w) \land f(\text{pred}(y), w, z) \]
Which Theories Support Recursion?

- PA
- Typed Finite Set Theory
- May take them as primitive
- With a universe we can have recursive type definitions
VIII. CORRECTNESS
• Specifications are more than stipulative definitions.
• In contrast to mathematical definitions, they are aimed at the construction of artefacts. They tell us what to build
• They give substance to the notions of correctness and malfunction. Here we deal with abstract artefacts
Correctness

S is correct relative to S (written R<S) iff

I. Dom(R) ⊆ Dom(S)

II. S(x, y) → R(x, y)
Where S=S↓ Dom(R)

- Where S is functional II reduces to: R(x, S(x)).
  It states that S correctly implements R
- R provides the correctness criteria for S
- Proof of correctness is a mathematical affair.
IX. COMPUTABLE SPECIFICATIONS
Must Specifications Be Computable?

- Do specifications need to be $\Sigma$?
- After all they are intended to be specifications of artefacts that have to be built/constructed?
- But there are delicate issues here.
- Even computable objects can have non-computable properties.
- For example a relation that has to be implemented has to be RE but we may require it to be functional and this is not part of the substance of the implementation; it is a property of it.
Implementable in Theory

- $S$ is **Strongly Implementable** iff $S$ is $\Sigma$
- $S$ is **Weakly Implementable** iff there exists a $\Sigma$-definition $R$ such that $S<R$.
- Halting Problem is not Implementable.
Interpretations in Arithmetic

- Theories of Data types need to have interpretations in Arithmetic
- The types need to be RE sets
- Specifications interpreted as RE relations
- Functions as at least semi-computable
X. AXIOMATIC DEFINITIONS
Implicit Definitions

- A term $R$ can be introduced axiomatically, i.e., via axioms of the expanded language $L^+$ (including $R$).
- Let $T^*$ be a set of sentences of the expanded language $L^+$ containing $R$. Then, $T^*$ constitutes an implicit (stipulative) definition of $R$.
- According to the traditional account it must meet the Conservativeness and Eliminability criteria. If it does meet these criteria, let us call $T^*$ admissible (for a definition of $R$).
- If it is admissible then it can be explicitly defined.
Axiomatic Notions

- With a rich enough base theory we may do things more explicitly
- Axiomatic definitions become standard schematic ones
- Set theory and algebraic structures
- E.g.
Groups

**Group**

\[
G : \text{Set}(U) \\
* : G \times G \Rightarrow G \\
e \in G
\]

\[
\forall g \in G \cdot \forall f \in G \cdot \forall h \in G \cdot g \ast (f \ast h) = (g \ast f) \ast h \\
\forall g \in G \cdot e \ast g = g \\
\forall f \in G \cdot \exists g \in G \cdot g \ast f = e
\]
Model Theoretic Account

- An interpretation $M^+$ of $L^+$ is an expansion of an interpretation $M$ of $L$ iff $M$ and $M^+$ have the same domain and they assign the same semantic values to the non-logical constants in $L$.
- $T^*$ is an implicit semantic definition of $R$ iff, for each interpretation $M$ of $L$, there is a unique model $M^+$ of $T^*$ such that $M^+$ is an expansion of $M$.
- If $T^*$ is admissible then $T^*$ is an implicit semantic definition of $R$.
- Beth's Definability Theorem. If $T^*$ is an implicit semantic definition of $R$ in a classical first-order language then $T^*$ is admissible.
More Details

- Computable Models