Representing Types in Map Theory

Sebastian Skalberg

skalberg@in.tum.de

Technische Universität München

Overview

- Introduction
- Simple Types
- Function Types
- Equality Types
- Classical Types

The Problem in a Nutshell

We want to be able to formalize statements like

> "for all booleans A, $A \vee \neg A$ is true"

and

> "when B is a natural number, $\gcd(B, B) = B$"

The Problem in a Nutshell

We want to be able to formalize statements like

> "for all booleans A, $A \vee \neg A$ is true"

and

> "when B is a natural number, $\gcd(B, B) = B$"

However, it is syntactically permissible to ask

> "what is the value of $\gcd(\mathsf{True}, \neg 2)$?"

in an untyped logic like map theory.

Some Observations

- In general, we cannot hope to have unique representations.

Some Observations

- In general, we cannot hope to have unique representations.
- Not all maps are meaningful for a given type: Assume $\bot$ and A were distinct representatives of some type, and that swap was given such that:

$$\mathsf{swap}(\bot) = A$$
$$\mathsf{swap}(A) = \bot$$

Then

$$A = \mathsf{swap}(\bot) \preceq \mathsf{swap}(A) = \bot$$

and hence $A = \bot$.

Some Observations

- In general, we cannot hope to have unique representations.
- Not all maps are meaningful for a given type: Assume $\bot$ and A were distinct representatives of some type, and that swap was given such that:

$$\mathsf{swap}(\bot) = A$$
$$\mathsf{swap}(A) = \bot$$

  Then

$$A = \mathsf{swap}(\bot) \preceq \mathsf{swap}(A) = \bot$$

  and hence $A = \bot$.
- Type membership is not decidable.

Desired Properties

- Type membership must be expressible within the theory and semi-decidable, ie., for each type α we want a function $\cdot$ **is** α which is true exactly when its argument is of type α and undefined otherwise.

Desired Properties

- Type membership must be expressible within the theory and semi-decidable, ie., for each type α we want a function $\cdot$ **is** α which is true exactly when its argument is of type α and undefined otherwise.
- The mapping from weak to strong representatives must be expressible within the theory, ie., we want a function $\approx_\alpha(\cdot)$ that takes its argument to the corresponding strong representative of the type α.

Definition: Simple Type

A *simple type* α is represented by a function, $\approx_\alpha(\cdot)$, satisfying

- $\approx_\alpha(\approx_\alpha(A)) = \approx_\alpha(A)$
- $\approx_\alpha(\bot) = \bot$

We say that a map A has type α if $\approx_\alpha(A)$ is defined. The function

$$A \textbf{ is } \alpha \stackrel{\text{def}}{=} \textbf{ifnil } \approx_\alpha(A) \textbf{ then True else True}$$

is true if A is of type α and undefined otherwise.

Example: Booleans

The type of booleans is described by the definitions

$$\begin{aligned} \mathsf{True} &\stackrel{\text{def}}{=} \mathsf{N} \\ \mathsf{False} &\stackrel{\text{def}}{=} \lambda x.\mathsf{N} \\ \approx_{\mathsf{bool}}(A) &\stackrel{\text{def}}{=} \textbf{ifnil } A \textbf{ then } \mathsf{True} \textbf{ else } \mathsf{False} \end{aligned}$$

Intuition: Typed Functions

When is $\mathcal{F}(\cdot)$ intuitively a function from α to β?

- If given something *not* of type α, it should be undefined, ie.,

$$A \textbf{ is } \alpha = \bot \quad \Longrightarrow \quad \mathcal{F}(A) = \bot$$

- If the function is total, we shall also want

$$A \textbf{ is } \alpha \quad \Longrightarrow \quad \mathcal{F}(A) \textbf{ is } \beta$$

- It should be well-defined, ie.,

$$\approx_\alpha(A) = \approx_\alpha(B) \quad \Longrightarrow \quad \approx_\beta(\mathcal{F}(A)) = \approx_\beta(\mathcal{F}(B))$$

Properties of Typed Functions

A function $\mathcal{F}(\cdot)$ is said to be of type $\alpha \Rightarrow \beta$ if

- $\mathcal{F}(\perp) = \perp$.
- If $\approx_\alpha(A) = \approx_\alpha(B)$ then $\mathcal{F}(A) = \mathcal{F}(B)$.
- For all maps A, $\approx_\beta(\mathcal{F}(A)) = \mathcal{F}(A)$.

Some Operations on Booleans

$$\neg A \stackrel{\text{def}}{=} \textbf{if } A \textbf{ then } \mathsf{False} \textbf{ else } \mathsf{True}$$

$$A \vee B \stackrel{\text{def}}{=} B \textbf{ is } \mathsf{bool} \mapsto \textbf{if } A \textbf{ then } \mathsf{False} \textbf{ else } \approx_{\mathsf{bool}}(B)$$

$$A \wedge B \stackrel{\text{def}}{=} B \textbf{ is } \mathsf{bool} \mapsto \textbf{if } A \textbf{ then } \approx_{\mathsf{bool}}(B) \textbf{ else } \mathsf{False}$$

$$A \to B \stackrel{\text{def}}{=} B \textbf{ is } \mathsf{bool} \mapsto \textbf{if } A \textbf{ then } \approx_{\mathsf{bool}}(B) \textbf{ else } \mathsf{True}$$

Strictness and Type Inference

The strictness of typed functions allows us for example to deduce

$$A \text{ is bool and } B \text{ is bool}$$

from the assumption

$$A \vee B \text{ is bool}$$

Especially, if disjunction was not strict, we could deduce nothing about B from the assumption True $\vee$ B. We can therefore state lemmas like

$$A \vee B \quad \Longrightarrow \quad B \vee A$$

without further assumptions.

Definition: Function Type

Whenever α and β, we define the *function type* $\alpha \Rightarrow \beta$ by

$$F \textbf{ is } \alpha \Rightarrow \beta \stackrel{\text{def}}{=} F \textbf{ is defined}$$

$$\approx_{\alpha \Rightarrow \beta}(F) \stackrel{\text{def}}{=} F \textbf{ is } \alpha \Rightarrow \beta \mapsto \lambda x.x \textbf{ is } \alpha \mapsto \approx_{\beta}(F`(\approx_{\alpha}(x)))$$

Note that this is the type of *partial* functions from α to β.

Example: Natural Numbers

The natural number n is represented strongly by the term

$$\lambda x_1 \ldots x_n.\mathsf{N}$$

The type of natural numbers is then described by the definitions

$$A \textbf{ is } \mathsf{nat} \stackrel{\text{def}}{=} \textbf{ifnil } A \textbf{ then } \mathsf{True} \textbf{ else } (A\text{`}\mathsf{N} \textbf{ is } \mathsf{nat})$$

$$\approx_{\mathsf{nat}}(A) \stackrel{\text{def}}{=} A \textbf{ is } \mathsf{nat} \mapsto \textbf{ifnil } A \textbf{ then } \mathsf{N} \textbf{ else } \mathsf{K}\text{`}(\approx_{\mathsf{nat}}(A\text{`}\mathsf{N}))$$

$$0 \stackrel{\text{def}}{=} \mathsf{N}$$

$$\mathsf{Suc}(A) \stackrel{\text{def}}{=} A \textbf{ is } \mathsf{nat} \mapsto \mathsf{K}\text{`}(\approx_{\mathsf{nat}}(A))$$

Example: Euclid's GCD Algorithm

Euclid's GCD algorithm is defined (almost) as usual:

$$\begin{aligned}\gcd(A,B) \stackrel{\text{def}}{=}\ & A \text{ is nat} \mapsto \\ & \quad \mathbf{case}\ B\ \mathbf{of}\ 0 \Rightarrow \approx_{\mathsf{nat}}(A) \\ & \qquad \mid \mathsf{Suc}\ x \Rightarrow \approx_{\mathsf{nat}}(\gcd(B, A\ \mathbf{mod}\ B))\end{aligned}$$

We can prove

$$\begin{aligned}\gcd(A,0) &= \approx_{\mathsf{nat}}(A) \\ \gcd(A,1) &= A \text{ is nat} \mapsto 1 \\ 0 <_{\mathsf{nat}} B \implies \gcd(A,B) &= \gcd(B, A\ \mathbf{mod}\ B)\end{aligned}$$

Definition: Equality Type

A simple type α is called an *equality type* if there exists a total function $=_\alpha$ of type $[\alpha, \alpha] \Rightarrow$ bool such that

- $=_\alpha$ is a equivalence relation on α.
- for all A and B, $A =_\alpha B$ implies $\approx_\alpha(A) = \approx_\alpha(B)$.

We shall say that $=_\alpha$ is *reflective* when the second property above holds.

Example: Addition

Since nat is an equality type, we can prove

$$A +_{\mathsf{nat}} B = B +_{\mathsf{nat}} A$$

by assuming A and B are natural numbers and then proving

$$\begin{array}{rl}
\forall y.A +_{\mathsf{nat}} y =_{\mathsf{nat}} y +_{\mathsf{nat}} A & \Longrightarrow \\
A +_{\mathsf{nat}} \approx_{\mathsf{nat}}(B) =_{\mathsf{nat}} \approx_{\mathsf{nat}}(B) +_{\mathsf{nat}} A & \Longrightarrow \\
A +_{\mathsf{nat}} B =_{\mathsf{nat}} B +_{\mathsf{nat}} A & \Longrightarrow \\
\approx_{\mathsf{nat}}(A +_{\mathsf{nat}} B) = \approx_{\mathsf{nat}}(B +_{\mathsf{nat}} A) & \Longrightarrow \\
A +_{\mathsf{nat}} B = B +_{\mathsf{nat}} A &
\end{array}$$

Definition: Classical Type

A simple type α is called a *classical type* if all strong representatives of α are classical, and there exists a function $\mathbf{CL}_\alpha(\cdot)$ such that

- $\mathbf{CL}_\alpha(\cdot)$ is a boolean predicate on the classical maps, i.e.,

$$A \text{ is classical} \implies \mathbf{CL}_\alpha(A) \text{ is bool}$$

- of the classical maps, only those of type α satisfy $\mathbf{CL}_\alpha(\cdot)$, i.e.,

$$A \text{ is classical and } \mathbf{CL}_\alpha(A) \implies A \text{ is } \alpha$$

- $\mathbf{CL}_\alpha(\cdot)$ holds for at least all strong α, i.e.,

$$A \text{ is } \alpha \implies \mathbf{CL}_\alpha(\approx_\alpha(A))$$

Typed Choice and Quantifiers

Using the $\mathbf{CL}_\alpha(\cdot)$ functions from before, we can now define typed choice operators and typed quantifiers:

$$\begin{aligned}
\varepsilon_\alpha\, x.A &\stackrel{\text{def}}{=} \varepsilon x.\mathbf{CL}_\alpha(x)\,\tilde{\wedge}\,A \\
\exists_\alpha\, x.A &\stackrel{\text{def}}{=} \exists x.\mathbf{CL}_\alpha(x)\,\tilde{\wedge}\,A \\
\forall_\alpha\, x.A &\stackrel{\text{def}}{=} \forall x.\mathbf{CL}_\alpha(x)\,\tilde{\rightarrow}\,A
\end{aligned}$$

where

$$\begin{aligned}
A\,\tilde{\wedge}\,B &\stackrel{\text{def}}{=} \textbf{if } A \textbf{ then } \approx_{\mathsf{bool}}(B) \textbf{ else } \mathsf{False} \\
A\,\tilde{\rightarrow}\,B &\stackrel{\text{def}}{=} \textbf{if } A \textbf{ then } \approx_{\mathsf{bool}}(B) \textbf{ else } \mathsf{True}
\end{aligned}$$

The Function Type Revisited

If α is a classical type and β is a simple type, we can now define the type β^α of total functions from α to β:

$$F \textbf{ is } \beta^\alpha \stackrel{\text{def}}{=} \forall_\alpha x.F`(\approx_\alpha(x)) \textbf{ is } \beta$$

$$\approx_{\beta^\alpha}(F) \stackrel{\text{def}}{=} F \textbf{ is } \beta^\alpha \mapsto \lambda x.x \textbf{ is } \alpha \mapsto \approx_\beta(F`(\approx_\alpha(x)))$$

Concluding Remarks

- The type schemes developed allow us to state and prove lemmas much like in syntactically comparable typed logics.
- While the class of classical maps is crucial for defining quantifiers over new types, classicality seldom turns up in proofs, once the basic properties of a new type has been proved.
- The notion of type has been almost indispensable for getting the Isabelle simplifier to work with map theory.

References

- Klaus Grue. Map theory with classical maps. Can be found at `http://www.diku.dk/users/grue`, 2001.
- Sebastian C. Skalberg. *An Interactive Proof System for Map Theory*. PhD thesis, University of Copenhagen, 2002.