10

PROGRAMMING IN THE SIMPLY TYPED λ-CALCULUS

This section explains how to use the simply typed λ-calculus to encode compound data structures and proofs in intuitionistic logic. We will use the interpreter as a typed language and, at the same time, as a proof assistant for the intuitionistic propositional logic.

This presentation of simply typed structures follows the Mikrokosmos tutorial and the previous sections on simply typed λ-calculus. All the code on this section is valid Mikrokosmos code.

10.1 FUNCTION TYPES AND TYPEABLE TERMS

Types can be activated with the commmand `:types on`. If types are activated, the interpreter will infer the principal type of every term before its evaluation. The type will then be displayed after the result of the computation.

Example 5 (Typed terms on Mikrokosmos). The following are examples of already defined terms on lambda calculus and their corresponding types. It is important to notice how our previously defined booleans have two different types; while our natural numbers will have all the same type except from zero, whose type is a generalization on the type of the natural numbers.

```
id    --- [1]: λa.a ⇒ id, I, ifelse :: A → A
true  --- [2]: λa.λb.a ⇒ K, true :: A → B → A
false --- [3]: λa.λb.b ⇒ nil, 0, false :: A → B → B
0     --- [4]: λa.λb.b ⇒ nil, 0, false :: A → B → B
1     --- [5]: λa.λb.(a b) ⇒ 1 :: (A → B) → A → B
2     --- [6]: λa.λb.(a (a b)) ⇒ 2 :: (A → A) → A → A
S     --- [7]: λa.λb.λc.((a c) (b c)) ⇒ S :: (A → B → C) → (A → B) → A → C
K     --- [8]: λa.λb.a ⇒ K, true :: A → B → A
```

If a term is found to be non-typeable, Mikrokosmos will output an error message signaling the fact. In this way, the evaluation of λ-terms that could potentially not

terminate is prevented. Only typed λ-terms will be evaluated while the option `:types` is on; this ensures the termination of every computation on typed terms.

Example 6 (Non-typeable terms on Mikrokosmos). Fixed point operators are a common example of non typeable terms. Its evaluation on untyped λ-calculus would not terminate; and the type inference algorithm fails on them.

```
fix
--- Error: non typeable expression
fix (\f.\n.iszero n 1 (plus (f (pred n)) (f (pred (pred n))))) 3
--- Error: non typeable expression
```

Note that the evaluation of compound λ-expressions where the fixpoint operators appear applied to other terms can terminate, but the terms are still non typeable.

10.2 PRODUCT, UNION, UNIT AND VOID TYPES

Until now, we have only used the function type. That is to say that we are working on the implicational fragment of the simply-typed lambda calculus we described on the first [BROKEN LINK: *Typing rules for the simply typed \lambda-calculus]. We are now going to extend our type system in the same sense we extended that simply-typed lambda calculus. The following types are added to the system

Type	Name	Description
$\to$	Function type	Functions from a type to another
$\times$	Product type	Cartesian product of types
$+$	Union type	Disjoint union of types
$\top$	Unit type	A type with exactly one element
$\bot$	Void type	A type with no elements

And the following typed constructors are added to the language,

Constructor	Type	Description
`(-,-)`	`A → B → A × B`	Pair of elements
`fst`	`(A × B) → A`	First projection
`snd`	`(A × B) → B`	Second projection
`inl`	`A → A + B`	First inclusion
`inr`	`B → A + B`	Second inclusion
`caseof`	`(A + B) → (A → C) → (B → C) → C`	Case analysis of an union
`unit`	`⊤`	Unital element
`abort`	`⊥ → A`	Empty function
`absurd`	`⊥ → ⊥`	Particular empty function

which correspond to the constructors we described on previous sections. The only new term is the `absurd` function, which is only a particular case of `abort`, useful when we want to make explicit that we are deriving an instance of the empty type. This addition will only make the logical interpretation on the following sections clearer.

Example 7 (Extended STLC on Mikrokosmos). The following are examples of typed terms and functions on Mikrokosmos using the extended typed constructors. The following terms are presented

- a function swapping pairs, as an example of pair types;
- two-case analysis of a number, deciding whether to multiply it by two or to compute its predecessor;
- difference between `abort` and `absurd`;
- example term containing the unit type.

```
:load types
swap = \m.(snd m,fst m)
swap
--- [1]: λa.((SND a),(FST a)) ⇒ swap :: (A × B) → B × A
caseof (inl 1) pred (mult 2)
caseof (inr 1) pred (mult 2)
--- [2]: λa.λb.b ⇒ nil, 0, false :: A → B → B
--- [3]: λa.λb.(a (a b)) ⇒ 2 :: (A → A) → A → A
\x.((abort x),(absurd x))
--- [4]: λa.((ABORT a),(ABSURD a)) :: ⊥ → A × ⊥
```

Now it is possible to define a new encoding of the booleans with an uniform type. The type `⊤ + ⊤` has two inhabitants, `inl ⊤` and `inr ⊤`; and they can be used by case analysis.

```
btrue = inl unit
bfalse = inr unit
bnot = \a.caseof a (\a.bfalse) (\a.btrue)
```

```
bnot btrue
--- [1]: (INR UNIT) ⇒ bfalse :: A + ⊤
bnot bfalse
--- [2]: (INL UNIT) ⇒ btrue :: ⊤ + A
```

With these extended types, Mikrokosmos can be used as a proof checker on first-order intuitionistic logic by virtue of the Curry-Howard correspondence.

10.3 A PROOF IN INTUITIONISTIC LOGIC

Under the logical interpretation of Mikrokosmos, we can transcribe proofs in intuitionistic logic to λ-terms and check them on the interpreter. The translation between logical propositions and types is straightforward, except for the **negation** of a proposition $\neg A$, that must be written as $(A \to \bot)$, a function to the empty type.

Theorem 8. *In intuitionistic logic, the double negation of the Law of Excluded Middle holds for every proposition, that is,*

$$\forall A\colon \neg\neg(A \vee \neg A)$$

Proof. Suppose $\neg(A \vee \neg A)$. We are going to prove first that, under this specific assumption, $\neg A$ holds. If A were true, $A \vee \neg A$ would be true and we would arrive to a contradition, so $\neg A$. But then, if we have $\neg A$ we also have $A \vee \neg A$ and we arrive to a contradiction with the assumption. We should conclude that $\neg\neg(A \vee \neg A)$. □

Note that this is, in fact, an intuitionistic proof. Although it seems to use the intuitionistically forbidden technique of proving by contradiction, it is actually only proving a negation. There is a difference between assuming A to prove $\neg A$ and assuming $\neg A$ to prove A: the first one is simply a proof of a negation, the second one uses implicitly the law of excluded middle.

This can be translated to the Mikrokosmos implementation of simply typed λ-calculus as the term

```
notnotlem = \f.absurd (f (inr (\a.f (inl a))))
notnotlem
--- [1]: λa.(ABSURD (a (INR λb.(a (INL b))))) :: ((A + (A → ⊥)) → ⊥) → ⊥
```

whose type is precisely $((\mathtt{A} + (\mathtt{A} \to \bot)) \to \bot) \to \bot$. The derivation tree can be seen directly on the interpreter as Figure 1 shows.

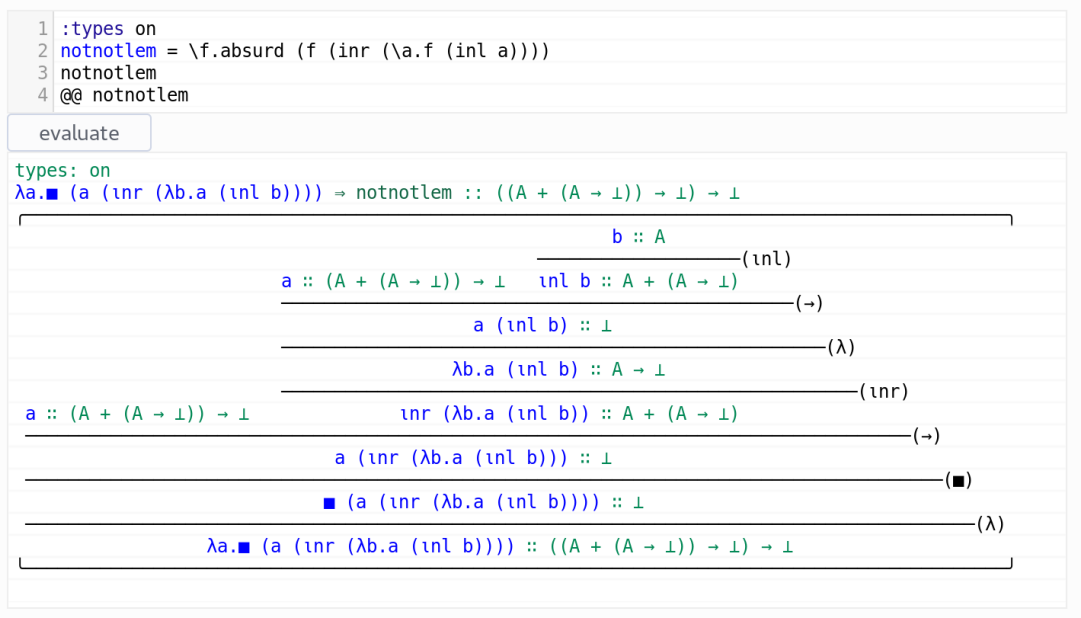

Figure 5: Proof of the double negation of LEM.