$\mathcal{T}IL$-Script

Functional Programming Based on Transparent Intensional Logic

Nikola Ciprich, Marie Duží, Michal Košinár

Technical University of Ostrava

December 13, 2007

The goals

- Design of TIL syntax and semantics suitable for a computer programming language
- Design of TIL inference engine
- Implementation of $\mathcal{T}IL$-Script as a free and open multiplatform language

Language of contructions

- Six kinds of constructions
- Two atomic:
  - Trivialisation
  - Variable
- and four molecular ones:
  - Closure
  - Composition
  - Execution
  - Double execution

Trivialisation

- Just returns an object
- Similar to pointer dereference
- TIL syntax: ${}^{0}Object$
- $\mathcal{T}IL$-Script syntax: ’Object

Variable

- TIL variable names: usually consist of lowercase letters
- $\mathcal{T}IL$-Script varible names:
  - begin with lowercase letters
  - can consist of letters, digits and underscore symbol

Closure

- Constructs an anonymous function f
- TIL syntax: $[\lambda x[{}^{0}+\ x\ {}^{0}1]]$
- $\mathcal{T}IL$-Script syntax: `[\x:Int ['+ x '1]]`

Creating functions using named closure

- To this end, we use the **def** keyword
- TIL syntax: *none*
- $\mathcal{T}IL$-Script syntax: `def Succ := [\x:Int ['+ x '1]].`

Composition

- The way of applying a function to its arguments
- TIL syntax: $[[\lambda x[^0+\ x\ ^0 1]\ ^0 5]$
- $\mathcal{T}IL$-Script syntax: `[[\x:Int ['+ x '1]] '5].`
- $\mathcal{T}IL$-Script syntax using named closure: `['Succ '5].`

Partiality

- Constructions can be *v*-improper
  (failing to *v*-construct anything)
- In principle improperness arises from composition:
  - partial function *f* undefined at its argument *a*
  - `['Plus [\x['Div '5 x]'0]'3]`
- Partiality "propagates" up

Types

- Basic:
  - **Bool** or **o** for truth values (o)
  - **Indiv** or **i** for individuals (ι)
  - **Time** or **t** for time (τ)
  - **Real** or **r** for real numbers
  - **Int** for integer numbers
  - **World** or **w** for possible worlds (ω)
- Functional (the same as in TIL):
  - (ttt) - the type of a binary function operating on real numbers
  - (ii)@wt - the type of an empirical function (attribute)
  - o@wt - the type of a proposition
  - etc
- Lists

Miscellaneous

- Quantifiers (**Exists**, **ForAll**, **Single**)
- Assigning values to dynamic variables using **let** keyword:
  - `let ind:='Berta.`
  - `let prop:=\w\t['Coming@wt 'Berta].`

Current state of the project

- The definition of syntax is almost finished
- Syntax analysis is almost implemented
- Inference machine (work in progress)

Questions, answers, comments

- Any questions?

Thank You

- Thank You