Functional Programming
Based on Transparent Intensional Logic

Nikola Ciprich, Marie Duží, Michal Košinár

VŠB—Technical University Ostrava
Faculty of Electrical Engineering and Computer Science
17. listopadu 15, 708 33, Ostrava-Poruba, Czech Republic
nikola.ciprich@linuxbox.cz

Abstract. In the paper we introduce an interpreter of the Transperent Intensional Logic (TIL), namely the TIL-Script language, its design and implementation. TIL is a brainchild of the Czech logician Pavel Tichý and in many ways a revolutionary logical system. The work on the TIL-Script project is being pursued by the team of undergraduate students. The paper we provide is a brief review of the TIL-Script syntax.

Key words: TIL, TIL-Script, Transparent Intensional Logic, syntax, grammar

1 Introduction

1.1 Motivation

Pavel Tichý set his work on TIL in 1970s, and since then he demonstrated its great expressive power, in particular in the area of analysis of natural language processing (see [3,4]). Since hitherto there is no computer implementation of TIL, we decided to create our own and make it available to public.

1.2 Objective

The notational syntax of TIL language of constructions as originally proposed by Tichý is not optimal for computer processing. It contains lots of non-ASCII characters which are difficult to type on a computer; for instance, Greek alphabet is used to denote particular atomic and molecular types, lots of upper indexing, etc. On the other hand, some special types that are useful from the computational point of view are missing (like the type for integers, a special type for time, lists and tuples, etc.). Therefore the changes in syntax and slight modifications in semantics had to be done in order that the code be easy to type and read. Since TIL as a higher-order logic is undecidable, it is also important to create a limited but working inference engine.

2 Constructions

There are six kinds of constructions in TIL, two atomic and four molecular ones. In the TIL-Script code each entry of a construction is terminated by a period.

2.1 Trivialisation

The first atomic construction is *trivialisation*. Given an object, it just returns the object. It could be thought of as compared to a pointer, and its dereference. In TIL, the Trivialisation is denoted by the ^{0} symbol (thus writing ^{0}*Object*). TIL-Script uses the apostrophe (′) symbol, as it's a symbol similar to the original one and easy to type. Thus we have the transcription of ^{0}*Object* into `'`Object. Any object, even a construction can be trivialised.

2.2 Variables

Another atomic construction is a variable. From the syntactic point of view, variables in TIL are usually named by lower-case letters; similarly in TIL-Script we can use any name beginning with a lowercase letter and consisting only of letters, numbers and '_' symbol.

2.3 Closures

The third construction is a *Closure* (also a *lambda closure*). It constructs an anonymous function f, that can then be applied to its argument a by using the Composition of the closure with a construction of a. In TIL, the syntax is similar to the lambda-calculus lambda abstraction, using the symbol λ to mark lambda-bound variables. For example, the Closure constructing the successor function can be written as $[\lambda x[^0{+}x^01]]$. TIL-Script retains this syntax, just replacing the symbol λ by '\' (and of course using the ′ symbol for trivialisation). Moreover, types have to be specified (more on types see below). So the TIL-Script notation of the *successor* closure is as follows:

```
[\x:Int ['+ x '1]].
```

Creating functions using named closures In TIL-Script, we can also construct a *named* function using Closure. To this end the d**ef** keyword is used; for instance, the above construction of the successor function named as *Succ* is in TIL-Script written as follows[1]:

```
def Succ := [\x:Int ['+ x '1]].
```

2.4 Compositions

The *Composition* construction is an instruction to apply a function to its arguments. There is no notational difference between TIL and TIL-Script concerning a Composition. For instance, here is a Composition of the above Closure with the Trivialisation of the number 5:

```
[[\x:Int ['+ x '1]] '5].
```

[1] Parameter types can also be specified in another way, this will be discussed later

Or, using the pre-defined name of the successor function, we have:

```
['Succ '5].
```

both constructing the number 6.

2.5 Double execution

Some constructions can be used twice over: the *Double Execution* of a construction X, in symbols ${}^2X$, v-constructs what is v-constructed by X. In TIL-Script the upper index ^{2} is replaced by the `^2` notation: `^2X`.

2.6 Partiality

It is important to note that TIL operates with partial functions. Thus a construction can fail to v-construct anything, i.e., it can be v-improper. This is in principle due to a Composition, when a partial function f is applied to an argument a but there is no value of f at a. Thus the Composition of the division function with an argument <x,0> encoded in TIL-Script by ['Div x '0], is v-improper for any x. And so is any Composition of another construction with the former, like, e.g., ['Plus ['Div x '0] '1]. We say that 'partiality is being strictly propagated up'.

Closure never fails to v-construct, it always v-constructs a function, even if it were a function that is undefined at all its arguments like `[\x ['Div x '0]]`.

3 Types

TIL defines four basic types: ι (iota) for the set of individuals (the universe of discourse), o (omicron) for the set of truth values, τ (tau) for the set of times and/or real numbers and ω (omega) for the set of possible worlds. TIL-Script slightly extends and modifies this set of basic types, as the mix of real numbers and times and the absence of integer numbers is not plausible. Therefore the types in TIL-Script are:

- `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

Special keyword **`Any`** stands for an unspecified type (α in TIL) that is used to indicate polymorphic function. The frequently used abbreviation of the Composition [[Cw]t] that is used for the extensionalisation of intensions (functions from possible worlds into chronologies of entities of a given type) is replaced by the `C@wt` notation.

3.1 Lists

From the computational point of view, an important type is that of a *list*. A list is a (potentially infinite) sequence of entities and can thus be constructed by a composed TIL construction. However, since lists (or tuples) are frequently used in a computer program, we decided to include list into TIL-Script as a special type. To this end we use the `l`**ist** keyword: `list`(type1 type2 ...). For example, the list of individuals is declared by `list(Indiv)` or `list(i)`, the list of triples of numbers is declared by `list(Int, Int, Int)`. Lists can also be defined recursively; thus by using `list(list(Int, Indiv))` we declare the list of lists of number-individual pairs. Types are then specified using the slash or colon (in Closures). There are two ways of defining types of function arguments. Either by defining the type of an entity (slash notation), or by defining the range of a variable (the colon notation). To adduce some examples, here are the types of individuals, empirical functions, properties and the range of variable `x`, respectively:

```
Charles, Tomas/Indiv.
Ostrava/Indiv.
President/(((ii)t)w).
President/(ii)@tw.
Property1/((oi)t)w).
Property2/(oi)@wt.
def Succ1:= [\x:Int ['+ x '1]].
Succ2/(Int Int).
def Succ2:=[\x ['+ x '1]].
```

4 Miscellaneous

4.1 Assignment operator

As an assignment operator, the `l`**et** keyword is used, e.g.: `let` city:='Ostrava. It assigns a value to a variable, and it is used mainly in recursive definitions and in the discourse processing (see [6]).

4.2 Quantifiers

For quantifiers we use the keywords `F`**orAll** (stands for ∀), `E`**xists** (stands for ∃), and `S`**ingle** for singularisers.

4.3 Infix × prefix notation

The first version of TIL-Script will support only prefix notation for all operators, further versions will also support the infix notation.

5 Examples

As an example we now introduce the transcription of the analysis of three agents communication (taken from [6]) into TIL-Script:

```
ind, loc/i.
pred, prof/(oi)@tw.
rel1/(oi(oi)@tw)@tw.
rel2/(oii)@tw.
rel3/(oio@tw)@tw.
prop/o@tw.
constr/*n.
```

Adam to *Cecil*: "Berta is comming. **She** is looking for a parking". 'Inform' message content:

```
\w\t['Comming@wt 'Berta].
```

Discourse variables updates:

```
let ind:='Berta.
let pred:='Coming.
let prop:=\w\t['Coming@wt 'Berta].

\w\t ^2['Sub ind 'she '['Looking_for@wt she 'Parking]].
                                 % (is transformed into:)
\w\t['Looking_for@wt 'Berta 'Parking].
```

Discourse variables updates:

```
let rel1:='Looking_for.
let pred:='Parking.
let prop:=\w\t['Looking_for@wt 'Berta 'Parking].
let prof:=\w\t\x['Looking_for@wt x 'Parking].
```

Cecil to *Adam*: "**So** am I". 'Inform' message content:

```
\w\t^2['Sub prof 'so '[so@wt 'Cecil]]. % transforms to:
\w\t['Looking_for@wt 'Cecil 'Parking].
```

Discourse variables updates:

```
let ind:='Cecil.
```

Adam to both: "There is a free parking at $p1$". 'Inform' message content:

```
\w\t[['Free 'Parking]@wt 'p1].
```

Discourse variables Updates:

```
let loc:='p1.
let pred:=['Free 'Parking].
let prop:=\w\t[['Free 'Parking]@wt 'p1].
```

Berta to *Adam*: "What do you mean by free parking?" 'Query' message content:

```
\w\t['Refine@wt '['Free 'Parking]].
```

Adam to *Berta*: "Free parking is a parking and some parts of it are not occupied". 'Reply' message content:

```
'['Free 'Parking] =
'[\w\t\x ['And
  ['Parking@wt x]
  ['Exists y [And
    ['Part_of@wt y x]
    Not ['Occupied@wt y]
  ]
]]].
```

6 Conclusion

In this paper we outlined the syntax of the TIL-Script language that is being developed within the project "Logic and Artificial intelligence for Multi-Agent Systems". TIL-Script is a FIPA compliant language in which the content of FCA messages is encoded. Agents communicate by messaging in TIL-Script.

In the paper we illustrated how smooth and natural the communication in TIL-Script is. The translation from natural language into TIL-Script messages (and vice versa) is near-to-isomorphic. Thus the humans can easily formulate the messages that the computational agents understand and perform.

Acknowledgements This research has been supported by the program "Information Society" of the Czech Academy of Sciences, project No. 1ET101940420 "Logic and Artificial Intelligence for Multi-Agent Systems"[2].

References

1. Duží M., Jespersen B., Materna P.: *Transparent Intensional Logic - Foundations and Applications*, mns. 2007.
2. Tichý P.: *The Foundations of Frege's Logic*, 1988, Berlin, New York: De Gruyter.
3. Tichý P.: *Cracking the Natural Language Code*, 1994 reprinted in [5]: pp. 843–857.
4. Tichý P.: *The Analysis of Natural Language*, 1994 reprinted in [5]: pp. 801–841.
5. *Pavel Tichý's Collected Papers in Logic and Philosophy* edited by V. Svoboda, B. Jespersen, C. Cheyne (eds.) Prague: Filosofia, Czech Academy of Sciences, and Dunedin: University of Otago Press.
6. Duží M.: *TIL as the Logic of Communication in a Multi-Agent System* Submitted to CICLING 2008.

[2] see `http://labis.vsb.cz`