Core-TyCO

Appendix to the Language Definition, yielding Version 0.2

Vasco T. Vasconcelos

DI–FCUL TR–01–5

July 2001

Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa
Campo Grande, 1749–016 Lisboa
Portugal

Technical reports are available at `http://www.di.fc.ul.pt/biblioteca/tech-reports`. The files are stored in PDF, with the report number as filename. Alternatively, reports are available by post from the above address.

Core-TyCO

Appendix to the Language Definition, yielding Version 0.2

Vasco T. Vasconcelos

July 2001

This is the third report on TyCO [2], a (still) experimental strongly and implicitly typed concurrent object based programming language based on a predicative polymorphic calculus of objects [1, 3], featuring asynchronous messages, objects, and procedures, together with a predicative polymorphic typing assignment system, assigning monomorphic types to variables and polymorphic types to procedures.

With respect to version 0.2, version 0.3 introduces

1. slight changes to the concrete syntax,
2. new derived constructors,
3. floating point numbers,
4. nested comments,
5. a means to split a program amongst different files, and
6. a clarification on the ambiguous nature of the syntax for processes.

Static and dynamic semantics of the language are not changed.

1 Syntax

1.1 Reserved words

Keywords `case`, `of`, and `/=` replace `switch`, `into`, and `<>`, respectively. New reserved words are: `tofloat`, `hd`, `tl`, `;`, and `:`.

1.2 Literals

An *integer constant* is given by the regular expression `0|[1-9]{digit}*`, were `{digit}` is the regular expression `[0-9]`.

Floating point constants are given by the regular expression `({integer}{digit}+)([eE][+-]?{digit}+)?`.

String constants are sequences of character enclosed between quotes (`"`). The following *escape sequences* are now supported: `\\`, `\'`, `\\\`, `\\n`, `\\t`, `\\a`, `\\b`, `\\f`, `\\r`, `\\v`.

1.3 Grammar

Different *bind*s are now longer separated by reserved word `and`. Comma does not separate *method*s. Syntactic categories for *bind* and *method* are now as follows.

multbind	::=	*bind* $\langle$*multbind*$\rangle$	multiple binding
methrow	::=	*method* $\langle$*methrow*$\rangle$	method row

1.4 Comments

TyCO now also permits *nested* comments, which start with {`-` and extend until the next unmatched `-`}.

1.5 Derived forms

The following syntactic category helps in defining the newly introduced derived constructors.

C	::=	a`!`$\langle$l$\rangle$	|	X	application

Below variables $z, z_1, \ldots, z_n$ are taken freshly, and ($n \geq 1$).

app`[`$\langle$*expseq*$\rangle$`]` `;` *proc*	$\Longrightarrow$	`new` z *app*`[`$\langle$*expseq*$\rangle z$`]` `|` *proc*`new` z
`(`$\langle$*varseq*$\rangle$`)` `:` *proc*	$\Longrightarrow$	`(`$\langle$*varseq*$\rangle z$`)` `:` z`! [ ] |` *proc*
`let` $varseq_1$ `=` app_1`[`$\langle expseq_1 \rangle$`]`	$\Longrightarrow$	`new` z_1, app_1`[`$\langle expseq_1$`,` $\rangle z_1$`] |` $\cdots$
$\cdots$ $varseq_n$ `=` app_n`[`$\langle expseq_n \rangle$`]`		`new` z_n app_n`[`$\langle expseq_n$`,` $\rangle z_n$`] |`
`in` *proc*		z_n`?(`$\langle varseq_n \rangle$`)` `=` $\cdots$
		z_1`?(`$\langle varseq_1 \rangle$`)` `=` *proc*

The underscore may be used in any *varseq*, including, for example, `let` processes.

1.6 Syntactic restrictions for derived forms

1. In `let` $varseq_1$ `=` $app_1\langle expseq_1\rangle \cdots varseq_n$ `=` $app_n\langle expseq_n\rangle$ `in` *proc*, the sequence of bindings $varseq_1 \ldots varseq_n$ may not contain the same variable twice.

1.7 File inclusion

The two tokens `include "aFile"` are replaced, at parsing time, by the contents of file `aFile`.

2 Static semantics

2.1 Types for primitive operations and objects

```
+         :  float float   →   float
-         :  float float   →   float
*         :  float float   →   float
/         :  float float   →   float
-         :  float         →   float

hd        :  string        →   string
tl        :  string        →   string

tofloat   :  int           →   float
```

Equality/inequality — `=`, `/=` — are defined on all types. Relational operators — `<`, `<=`, `>`, `>=` — available also for floating point numbers.

The basic stream based I/O primitive object gets a six new methods; three of them change type.

```
io  :  {getb : () → bool
        putb : bool → ()
        printb : bool
        geti : () → int
        puti : int → ()
        printi : int
        gets : () → string
        puts : string → ()
        prints : string
        getf : () → float
        putf : float → ()
        printf : float}
```

References

[1] Vasco T. Vasconcelos. A predicative polymorphic type for a calculus of objects. In *Type Theory and its Applications to Computer Systems*, number 851 in RIMS Lecture Notes, pages 78–87. Kyoto University, July 1993.

[2] Vasco T. Vasconcelos and Rui Bastos. Core-TyCO, the language definition, version 0.1. DI/FCUL TR 98–3, Department of Informatics, Faculty of Sciences, University of Lisbon, March 1998.

[3] Vasco T. Vasconcelos and Mario Tokoro. A typing system for a calculus of objects. In *1st International Symposium on Object Technologies for Advanced Software*, volume 472 of *LNCS*, pages 460–474. Springer-Verlag, November 1993.