SIMPLE — Typed — Dynamic

Grigore Roşu and Traian Florin Şerbănuţă (grosu,tserban2@illinois.edu)
University of Illinois at Urbana-Champaign

Abstract

This is the K dynamic semantics of the typed SIMPLE language. It is very similar to the semantics of the untyped SIMPLE, the difference being that we now dynamically check the typing policy described in the static semantics of typed SIMPLE. Because the static semantics of typed SIMPLE ensures that all expressions have a unique type, we can use the same syntax for both semantics, such as semantic rules that are accessing an array cell or a loop. We will highlight the differences between the two semantics and show that the untyped SIMPLE is an extend with the semantics. We recommend the reader to consult the Typed-Static and the untyped-Static semantics of SIMPLE before reading this semantics.

```
module SIMPLE-TYPED-DYNAMIC-SYNTAX
```

Syntax

The syntax of typed SIMPLE extends that of untyped SIMPLE with support for declaring types to variables and functions.

The syntax below is identical to that of the static semantics of typed SIMPLE. However, the K attributes attributes are like those of the untyped SIMPLE, to support the desired evaluation strategies of the various language constructs.

```
syntax K ::= main
```

Types

```
syntax Type ::= int#
            | bool#
            | void#
            | array#
            | fun#
            | Type "*" Type
            | Type "->" Type
            | "(" Type ")"

syntax Types ::= List{Type, ","}
```

Declarations

```
syntax Param ::= Type Id

syntax Params ::= List{Param, ","}

syntax Decl ::= Type Exp;
            | Type Id(Params){Block}
```

Expressions

```
syntax Exp ::= Int
             | Bool
             | Id
             | Id"["Exp"]"#strict(1,2)
             | "-"Exp#strict
             | "!"Exp#strict
             | Exp"*"Exp#strict(1,2)
             | Exp"/"Exp#strict(1,2)
             | Exp"+"Exp#strict(1,2)
             | Exp"-"Exp#strict(1,2)
             | Exp"<"Exp#strict(1,2)
             | Exp"<="Exp#strict(1,2)
             | Exp">"Exp#strict(1,2)
             | Exp">="Exp#strict(1,2)
             | Exp"==="Exp#strict(1,2)
             | Exp"!="Exp#strict(1,2)
             | Exp"&&"Exp#strict(1,2)
             | Exp"||"Exp#strict(1,2)
             | Exp"?"Exp":"Exp#strict(1,2,3)
             | Id"="Exp#strict(2)
             | Id"["Exp"]"="Exp#strict(1,2,3)
             | "("Exp"->"Exp")"
             | "("Exp")"
             | "("Exp"*"Exp")"
             | "("Exp"/"Exp")"
             | "("Exp"+"Exp")"
             | "("Exp"-"Exp")"
             | "("Exp"<"Exp")"
             | "("Exp"<="Exp")"
             | "("Exp">"Exp")"
             | "("Exp">="Exp")"
             | "("Exp"==="Exp")"
             | "("Exp"!="Exp")"
             | "("Exp"&&"Exp")"
             | "("Exp"||"Exp")"
             | "("Exp"?"Exp":"Exp")"
             | "("Id"="Exp")"
             | "("Id"["Exp"]"="Exp")"
             | Id"("Exps")"#strict(1)
             | "("Id"("Exps")")"
             | "("Exp")"
             | "("Exp"*"Exp")"
             | "("Exp"/"Exp")"
             | "("Exp"+"Exp")"
             | "("Exp"-"Exp")"
             | "("Exp"<"Exp")"
             | "("Exp"<="Exp")"
             | "("Exp">"Exp")"
             | "("Exp">="Exp")"
             | "("Exp"==="Exp")"
             | "("Exp"!="Exp")"
             | "("Exp"&&"Exp")"
             | "("Exp"||"Exp")"
             | "("Exp"?"Exp":"Exp")"
             | "("Id"="Exp")"
             | "("Id"["Exp"]"="Exp")"
             | "("Id"("Exps")")"
             | "("Exp")"
```

Like in the static semantics, there is no need for lists of identifiers (because we now have lists of parameters).

```
syntax Exps ::= List{Exp, ","}#strict(1)
```

Statements

```
syntax Block ::= {}
               | {Stmts}

syntax Stmt ::= Decl
              | Block
              | Exp;#strict(1)
              | if(Exp)Block else Block#strict(1,2,3)
              | if(Exp)Block
              | while(Exp)Block
              | for(Decl;Exp;Exp)Block
              | return;
              | return Exp;#strict(1)
              | assert(Exp);#strict(1)
              | assert(Exp,Exp);#strict(1,2)
              | print(Exp);#strict(1)
              | read(Id);
              | Id=Exp;#strict(2)
              | Id[Exp]=Exp;#strict(1,2,3)
              | Id=Exp[Exp];#strict(2,1)
              | Id=Exp[Exp][Exp];#strict(2,1,3)
              | Id=Id(Exps);#strict(2)
              | Id=Id(Id(Exps));#strict(2)
              | Id[Exp]=Id(Exps);#strict(1,2)
              | Id[Exp]=Id(Id(Exps));#strict(1,2)
              | Id=Exp[Exp];#strict(2,1)
              | Id=Exp[Exp][Exp];#strict(2,1,3)
              | Id=Id(Exp);#strict(2)
              | Id=Id(Id(Exp));#strict(2)
              | Id[Exp]=Id(Exp);#strict(1,2)
              | Id[Exp]=Id(Id(Exp));#strict(1,2)
              | Id=Exp[Exp];#strict(2,1)
              | Id=Exp[Exp][Exp];#strict(2,1,3)
              | Id=Exp[Exp];#strict(2,1)
              | Id=Exp[Exp][Exp];#strict(2,1,3)
```

Like in the static semantics, there is no need for lists of identifiers (because we now have lists of parameters).

```
syntax Stmts ::= List{Stmt, ""}#strict(1)
```

Statements

```
syntax Block ::= {}
               | {Stmts}

syntax Decl ::= int#
              | bool#
              | void#
              | int#Id#Id#Id#Int#Int#Int
```