Implementing the add() protocol

John Doe

July 17, 2017

1 Introduction

We start by writing syntax, type checking rules and the operational semantics of the *add()* protocol.

2 Syntax

⟨*program*⟩ ::= '`protocol (`'⟨*args*⟩'`) : party`' ⟨*Idlist*⟩ '{' ⟨*body*⟩ '}'

⟨*args*⟩ ::= ⟨*arg*⟩
| ⟨*arg*⟩,⟨*args*⟩

⟨*arg*⟩ ::= ⟨τ⟩ ⟨*Id*⟩

⟨τ⟩ ::= '`Int`'
| '`Field`'

⟨*Idlist*⟩ ::= ⟨*Id*⟩
| ⟨*Id*⟩,⟨*Idlist*⟩

⟨*Id*⟩ ::= '`[a-z, A-Z]`$^{+}$'
| ⟨*Id*⟩.⟨*Id*⟩
| '`environment`'

⟨*body*⟩ ::= ⟨*statement*⟩$^{+}$

⟨*statement*⟩ ::= ⟨*pvtstmt*⟩
| ⟨*fwdstmt*⟩
| ⟨*assgnmntstmt*⟩
| ⟨*declstmt*⟩

⟨*pvtstmt*⟩ ::= '`in`' ⟨*Id*⟩ '{' ⟨*statment*⟩ '}'

⟨*fwdstmt*⟩ ::= ⟨*Id*⟩ '⇒' ⟨*Id*⟩

⟨*assgnmntstmt*⟩ ::= ⟨*Id*⟩ '=' ⟨*exp*⟩

⟨*exp*⟩ ::= n
| ⟨*Id*⟩
| ⟨*exp*⟩ '+' ⟨*exp*⟩
| '('⟨*exp*⟩')'
| ⟨*exp*⟩ '*' ⟨*exp*⟩

⟨*declstmt*⟩ ::= ⟨τ⟩ ⟨*Id*⟩

⟨*Γ*⟩ ::= ⟨*Id*⟩ → (⟨τ⟩, ⟨*partyScope*⟩)

⟨*partyScope*⟩ ::= ⟨ζ⟩
| ⟨*Id*⟩

3 Type checking

The following rules states how the types checks are going to be performed

3.1 Type Environment

The *type* environment of Cooler consists of three parts.

- A type environment that contains all possible scopes denoted by A
- Γ that maps Identifier to a tuple *(type, partyScope)*. Here *type* denotes the type of the Identifier while *partyScope* denotes a scope the Identifier belongs to. A *partyScope* is the scope local to each party.
  $\Gamma(Id) \rightarrow$ (*type, partyScope*)
- s denotes the current scope

3.2 Type Checking Rules

The notation used for each of the type checking rule is as follows:

$$\frac{\vdots}{A, \Gamma, s \vdash e : T}$$

here A, Γ and s represents the three type environments respectively as introduced above and the expression e evaluates to type T.

$$\frac{(Id, s) \notin \Gamma}{A, \Gamma, s \vdash \tau Id : \tau} \text{ [declstmt]}$$

$$\frac{}{A, \Gamma, s \vdash n : Int} \text{ [Int]}$$

Here x is a special Int that is not of the form *Id.Id*

$$\frac{\Gamma(x) = Int}{A, \Gamma, s \vdash x : Int}\ [\text{x}]$$

$$\frac{\begin{array}{l} A, \Gamma, s \vdash e_1 : Int \\ A, \Gamma, s \vdash e_2 : Int \\ op \in \{+, *\} \end{array}}{A, \Gamma, s \vdash e_1\ op\ e_2 : Int}\ [\text{exp op exp}]$$

$$\frac{A, \Gamma, s \vdash e_1 : Int}{A, \Gamma, s \vdash (e_1) : Int}\ [(\text{exp})]$$

$$\frac{\begin{array}{l} \Gamma, s \vdash e : T \\ \Gamma(x) = (T, s) \end{array}}{A, \Gamma, s \vdash x = e :\ (\ )}[\text{assgnmntstmt}]$$

x and *environment* is always well typed

$$\frac{A(x)}{A, \_, \_ \vdash x : ()}\ [\text{x}]$$

$$\frac{}{\_, \_, \_ \vdash environment : ()}\ [\text{environment}]$$

$$\frac{\begin{array}{l} A, \Gamma, s \vdash Id_1 : T \\ A, \Gamma, s \vdash Id_2 : T \end{array}}{A, \Gamma, s \vdash Id_1 \Rightarrow Id_2 : ()}\ [\text{fwdstmt Id to Id}]$$

$$\frac{A, \Gamma, s \vdash Id : T}{A, \Gamma, s \vdash Id \Rightarrow environment : ()}[\text{fwdstmt Id to env}]$$

$$\frac{A, \Gamma, s \vdash Id : T}{A, \Gamma, s \vdash environment \Rightarrow Id : ()}[\text{fwdstmt env to Id}]$$

$$\frac{A, \Gamma, x \vdash B : ()}{A, \Gamma, \zeta \vdash \text{in } x\{B\} : ()}\ [\text{pvtstmt}]$$

$$\frac{\begin{array}{l} A, \_, \_ \vdash x : () \\ A, \Gamma, x \vdash y : T \end{array}}{A, \Gamma, \zeta \vdash x.y : T}\ [\text{x.y}]$$

$$\frac{\{\mathscr{G}, scopelist\}, f(args), \zeta \vdash B : ()}{\{\mathscr{G}\}, \phi \vdash protocol(args) : \text{party } scopelist\{B\} : ()}\ [\text{proto-col}]$$

4 Operational Rules

The following section describes the operational semantics of our language: The notation used is as follows.

$$\frac{\vdots}{S, E, s \vdash e : v, S'}$$

$$\frac{\begin{array}{c} E(Id) = l \\ S(l) = v \end{array}}{S, E, s \vdash Id : v, S}[\text{Var}]$$