Syntax for VOSENID

Julien Lange
(based on the documentation generated by the BNF-Converter)

September 23, 2010

This document presents the syntax accepted by VOSENID as *Global Types*, *Process* and *Local Types*. Section 1 gives the common syntax, used in the three languages (i.e. reserved words, expressions, etc). Section 2 gives the syntax for global types, Section 3 the syntax for processes and Section 4 the syntax for local types[1].

1 Common Syntax

1.1 Reserved words and symbols

The set of reserved words is the set of terminals appearing in the grammar. Those reserved words that consist of non-letter characters are called symbols, and they are treated in a different way from those that are similar to identifiers. The lexer follows rules familiar from languages like Haskell, C, and Java, including longest match and spacing conventions.

The reserved words used in VOSENID are the following:

`Exists`	`False`	`Forall`
`Global`	`Process`	`True`
`and`	`bool`	`date`
`else`	`end`	`if`
`init`	`int`	`join`
`mu`	`not`	`nu`
`or`	`string`	`then`

The symbols used in VOSENID are the following:

`:`	`[`	`]`
`{`	`}`	`|`
`->`	`(`	`)`
`.`	`::`	`,`
`!`	`;`	`((`
`))`	`?`	`$`
`&`	`-`	`/`
`==`	`!=`	`>`
`<`	`<=`	`>=`
`=>`	`_|_`	`@`

1.2 Comments

Single-line comments begin with `//`, `--`.
There are no multiple-line comments in the grammar.

[1] One may have to give local types as input when write protocols in which participants delegate their session to others.

1.3 Objects

Identifiers ⟨*Ident*⟩ are unquoted strings beginning with a letter, followed by any combination of letters, digits, and the characters `_ '`, reserved words excluded. In the following, ⟨*NonTerm+*⟩ means a list of object *NonTerm*, separated by "," and containing at least one element. ⟨*NonTerm**⟩ is similar but may be empty.

⟨*Label*⟩ ::= ⟨*Ident*⟩

⟨*Variable*⟩ ::= ⟨*Ident*⟩

⟨*VarDecl*⟩ ::= ⟨*Variable*⟩ `:` ⟨*Typ*⟩

⟨*Participant*⟩ ::= ⟨*Ident*⟩

⟨*BranchId*⟩ ::= ⟨*Ident*⟩

⟨*Channel*⟩ ::= ⟨*Ident*⟩

⟨*DelegDecl*⟩ ::= `[` ⟨*Channel+*⟩ `]` `:` ⟨*LocalType*⟩ `@` ⟨*Participant*⟩

⟨*Typ*⟩ ::= `int`
| `bool`
| `string`
| `date`

1.4 Expressions

Integer literals ⟨*Int*⟩ are nonempty sequences of digits.

String literals ⟨*String*⟩ have the form `"`x`"`, where x is any sequence of any characters except `"` unless preceded by `\`.

⟨*Exp*⟩ ::= ⟨*Integer*⟩
| ⟨*Variable*⟩
| ⟨*String*⟩
| ⟨*Integer*⟩ `/` ⟨*Integer*⟩ `/` ⟨*Integer*⟩
| `True`
| `False`
| `not` ⟨*Exp1*⟩
| `(` ⟨*Exp*⟩ `)`
| ⟨*Exp*⟩ ⟨*BinOp*⟩ ⟨*Exp1*⟩
| `Exists` ⟨*VarDecl+*⟩ `(` ⟨*Exp*⟩ `)`
| `Forall` ⟨*VarDecl+*⟩ `(` ⟨*Exp*⟩ `)`

⟨*BinOp*⟩ ::= `or`
| `and`
| `==`
| `!=`
| `>`
| `<`
| `<=`
| `>=`
| `=>`

⟨*Assertion*⟩ ::= ⟨*Exp*⟩
| –

2 Syntax of Global Types

Non-terminals are enclosed between ⟨ and ⟩. The symbols ::= (production), | (union) and ϵ (empty rule) belong to the BNF notation. All other symbols are terminals.

⟨*Program*⟩ ::= ⟨*GlobalSpecification+*⟩
| `Process :` ⟨*Process*⟩
| ⟨*GlobalSpecification+*⟩ `Process :` ⟨*Process*⟩

⟨*GlobalSpecification*⟩ ::= `Global [` ⟨*Channel*⟩ `] :` ⟨*DistributedInteraction*⟩

⟨*DistributedInteraction*⟩ ::= ⟨*Interaction*⟩
| { ⟨*ListConcurInteraction*⟩ }

⟨*ListConcurInteraction*⟩ ::= ⟨*DistributedInteraction*⟩ `|` ⟨*ListConcurInteraction*⟩
| ⟨*DistributedInteraction*⟩

⟨*Interaction*⟩ ::=
| ⟨*Participant*⟩ `->` ⟨*Participant*⟩ `:` ⟨*Channel*⟩ `(` ⟨*VarDecl+*⟩ `) [` ⟨*Assertion*⟩ `] .` ⟨*DistributedInteraction*⟩
| ⟨*Participant*⟩ `->` ⟨*Participant*⟩ `:` ⟨*Channel*⟩ `(` ⟨*DelegDecl*⟩ `) [` ⟨*Assertion*⟩ `] .` ⟨*DistributedInteraction*⟩
| ⟨*Participant*⟩ `->` ⟨*Participant*⟩ `:` ⟨*Channel*⟩ `::` ⟨*BranchId*⟩ { ⟨*BranchElmt+*⟩ }
| `end`
| `mu` ⟨*Variable*⟩ `(` ⟨*Exp**⟩ `) (` ⟨*VarDeclLoc**⟩ `) [` ⟨*Assertion*⟩ `] .` ⟨*DistributedInteraction*⟩
| ⟨*Variable*⟩ `(` ⟨*Exp**⟩ `)`

⟨*BranchElmt*⟩ ::= `[` ⟨*Assertion*⟩ `]` ⟨*Label*⟩ `:` ⟨*DistributedInteraction*⟩

⟨*VarDeclLoc*⟩ ::= ⟨*VarDecl*⟩ `@` ⟨*Location*⟩

⟨*Location*⟩ ::= { ⟨*Participant*⟩ `,` ⟨*Participant*⟩ }

3 Syntax of Processes

⟨*Process*⟩ ::= ⟨*Primitive*⟩
| { ⟨*ListConcurProcess*⟩ }

⟨*ListConcurProcess*⟩ ::= ⟨*Process*⟩ `|` ⟨*ListConcurProcess*⟩
| ⟨*Process*⟩

⟨*Primitive*⟩ ::= `init :` ⟨*Channel*⟩ `[` ⟨*Participant*⟩`,`⟨*Participant+*⟩ `] (` ⟨*Channel+*⟩ `) .` ⟨*Process*⟩
| `join :` ⟨*Channel*⟩ `[` ⟨*Participant*⟩ `] (` ⟨*Channel+*⟩ `) .` ⟨*Process*⟩
| ⟨*Channel*⟩ `!` `(` ⟨*Exp+*⟩ `) (` ⟨*VarDecl+*⟩ `) [` ⟨*Assertion*⟩ `] ;` ⟨*Process*⟩
| ⟨*Channel*⟩ `!` `((` ⟨*Channel+*⟩ `)) (` ⟨*DelegDecl*⟩ `) [` ⟨*Assertion*⟩ `] ;` ⟨*Process*⟩
| ⟨*Channel*⟩ `?` `(` ⟨*VarDecl+*⟩ `) [` ⟨*Assertion*⟩ `] ;` ⟨*Process*⟩
| ⟨*Channel*⟩ `?` `((` ⟨*DelegDecl*⟩ `))` `[` ⟨*Assertion*⟩ `] ;` ⟨*Process*⟩
| ⟨*Channel*⟩ `$ [` ⟨*Assertion*⟩ `]` ⟨*BranchId*⟩ `.` ⟨*Label*⟩ `;` ⟨*Process*⟩
| ⟨*Channel*⟩ `&` ⟨*BranchId*⟩ { ⟨*PBranchElmt+*⟩ }
| `end`
| `if` ⟨*Exp*⟩ `then` ⟨*Process*⟩ `else` ⟨*Process*⟩
| `(` `nu` ⟨*Channel*⟩ `)` ⟨*Process*⟩
| `mu` ⟨*Variable*⟩ `(` ⟨*Exp**⟩ `;` ⟨*LLChannel*⟩ `) (` ⟨*VarDecl**⟩ `;` ⟨*ChannelList*⟩ `) .` ⟨*Process*⟩
| ⟨*Variable*⟩ `(` ⟨*Exp**⟩ `;` ⟨*LLChannel*⟩ `)`

⟨*PBranchElmt*⟩ ::= `[` ⟨*Assertion*⟩ `]` ⟨*Label*⟩ `:` ⟨*Process*⟩

⟨*LLChannel*⟩ ::= ⟨*Channel+*⟩ `;` ⟨*LLChannel*⟩
| ⟨*Channel+*⟩

4 Syntax for Local Types

⟨*LocalType*⟩ ::= ⟨*Channel*⟩ `!` `<` ⟨*VarDecl+*⟩ `>` `[` ⟨*Assertion*⟩ `]` `;` ⟨*LocalType*⟩
| ⟨*Channel*⟩ `!` `<` ⟨*DelegDecl*⟩ `>` `[` ⟨*Assertion*⟩ `]` `;` ⟨*LocalType*⟩
| ⟨*Channel*⟩ `?` `<` ⟨*VarDecl+*⟩ `>` `[` ⟨*Assertion*⟩ `]` `;` ⟨*LocalType*⟩
| ⟨*Channel*⟩ `?` `<` ⟨*DelegDecl*⟩ `>` `[` ⟨*Assertion*⟩ `]` `;` ⟨*LocalType*⟩
| ⟨*Channel*⟩ `&` ⟨*BranchId*⟩ `{` ⟨*LabelElmt+*⟩ `}`
| ⟨*Channel*⟩ `$` ⟨*BranchId*⟩ `{` ⟨*LabelElmt+*⟩ `}`
| `mu` ⟨*Variable*⟩ `(` ⟨*Exp**⟩ `)` `(` ⟨*VarDecl**⟩ `)` `[` ⟨*Assertion*⟩ `]` `.` ⟨*LocalType*⟩
| ⟨*Variable*⟩ `(` ⟨*Exp**⟩ `)`
| `end`
| `_|_`

⟨*LabelElmt*⟩ ::= `[` ⟨*Assertion*⟩ `]` ⟨*Label*⟩ `:` ⟨*LocalType*⟩