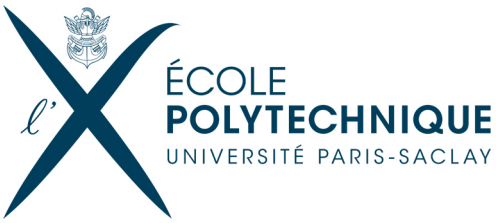
ÉCOLE POLYTECHNIQUE
PROMOTION X2011
Férey Gaspard



RAPPORT DE STAGE D'OPTION SCIENTIFIQUE

Titre

NON CONFIDENTIEL

Option :	INFORMATIQUE
Champ de l'option :	Math-Informatique
Directeur de l'option :	Olivier Bournez
Directeur de stage :	Olivier Bournez
Dates du stage :	7 avril - 22 aout 2014
Nom et adresse de l'organisme :	SRI International Computer Science Laboratory (CSL) 333 Ravenswood Avenue Menlo Park, CA 94025-3493 United States

June 16, 2014

Contents

1 Introduction

2 PVS

3 Translating PVS to C

We only convert a fragment of PVS.

4 PVS Syntax

We describe here the syntax of PVS and the objects system used to represent them in Lisp. Some slots of the classes are voluntarily omitted. For a full description of PVS parser representation, refer to [**?**].

Expr ::= *Number*
| *String*
| *Name*
| *Id* `!` *Number*
| *Expr Arguments*
| *Expr Binop Expr*
| *Unaryop Expr*
| *Expr* ‘ –*Id* | *Number* ″
| `(` *Expr*$^{+}_{,}$ `)`
| `(:` *Expr*$^{*}_{,}$ `:)`
| `[|` *Expr*$^{*}_{,}$ `|]`
| `(|` *Expr*$^{*}_{,}$ `|)`
| `{|` *Expr*$^{*}_{,}$ `|}`
| `(#` *Assignment*$^{+}_{,}$ `#)`
| *Expr* `::` *TypeExpr*
| *IfExpr*
| *BindingExpr*
| `{` *SetBindings* `|` *Expr* `}`
| `LET` *LetBinding*$^{+}_{,}$ `IN` *Expr*
| *Expr* `WHERE` *LetBinding*$^{+}_{,}$
| *Expr* `WITH [` *Assignment*$^{+}_{,}$ `]`
| `CASES` *Expr* `OF` *Selection*$^{+}_{,}$ [`ELSE` *Expr*] `ENDCASES`
| `COND` { *Expr* `->` *Expr*}$^{+}_{,}$ [`,` `ELSE ->` *Expr*] `ENDCOND`
| *TableExpr*

IfExpr ::= `IF` *Expr* `THEN` *Expr*
`{ELSIF` *Expr* `THEN` *Expr*`}` * `ELSE` *Expr* `ENDIF`

BindingExpr ::= *BindingOp* *LambdaBindings* `:` *Expr*

BindingOp ::= `LAMBDA` | `FORALL` | `EXISTS` | `{` *IdOp* `!` `}`

LambdaBindings ::= *LambdaBinding* `[` `[,]` *LambdaBindings*`]`

LambdaBinding ::= *IdOp* | *Bindings*

SetBindings ::= *SetBinding* `[` `[,]` *SetBindings*`]`

SetBinding ::= `{`*IdOp* `[:` *TypeExpr*`]` `}` | *Bindings*

Assignment ::= *AssignArgs* `{:=` | `|->}` *Expr*

AssignArgs ::= *Id* `[!` *Number*`]`
| *Number*
| *AssignArg*$^+$

AssignArg ::= `(` *Expr*$^+_,$ `)`
| `‘` *Id*
| `‘` *Number*

Selection ::= *IdOp* `[(` *IdOps* `)]` `:` *Expr*

TableExpr ::= `TABLE` `[`*Expr*`]` `[,` *Expr*`]`
`[`*ColHeading*`]`
TableEntry$^+$ `ENDTABLE`

ColHeading ::= `|[` *Expr* `{|` `{`*Expr* | `ELSE``}` `}` $^+$ `]|`

TableEntry ::= `{|` `[`*Expr* | `ELSE``]` `}` $^+$ `||`

LetBinding ::= `{` *LetBind* | `(` *LetBind*$^+_,$ `)` `}` `=` *Expr*

LetBind ::= *IdOp* *Bindings** `[:` *TypeExpr*`]`

Arguments ::= `(` *Expr*$^+_,$ `)`

5 Types

A PVS theory can be typechecked using the emacs interface `M-x typecheck` or with Lisp function `(tc name-theory)`. This first runs the PVS parser on the code and generates CLOS objects to represent it. Then, the PVS typechecker is run on this internal representation of the theory and tries to give a type to all expressions generating TCC when needed.

Here we describe how PVS types are represented in Lisp. The syntax of PVS we allow

TypeExpr ::= *Name*
| *EnumerationType*
| *Subtype*
| *TypeApplication*
| *FunctionType*
| *TupleType*
| *CotupleType*
| *RecordType*

EnumerationType ::= { *IdOps* }

Subtype ::= { *SetBindings* | *Expr* }
| (*Expr*)

TypeApplication ::= *Name Arguments*

FunctionType ::= [FUNCTION | ARRAY]
[-[*IdOp* :] *TypeExpr* ″$^{+}_{,}$ -> *TypeExpr*]

TupleType ::= [-[*IdOp* :] *TypeExpr* ″$^{+}_{,}$]

CotupleType ::= [-[*IdOp* :] *TypeExpr* ″$^{+}_{+}$]

RecordType ::= [# *FieldDecls*$^{+}_{,}$ #]

FieldDecls ::= *Ids* : *TypeExpr*

type-expr ⊂ syntax [*abstract class*]

type-name ⊂ type-expr name [*class*]
adt

subtype ⊂ type-expr [*class*]
supertype
predicate

funtype ⊂ type-expr [*class*]
domain
range.

tupletype ⊂ type-expr [*class*]
types

recordtype ⊂ type-expr [*class*]
fields

boolean, number, number_field, real, rational, integer, $A \to B$, restricted types below(10) := $\{x : \text{int} | 0 \leq x < 10\}$) enum datatype

C types: [unsigned] char, int, long, double boolean arrays strings enum struct and others: short int, float, union, size_t, ...

5.1 Fragment of PVS syntax

5.2 Difficulties

if-expr update-expr

6 Other activities at SRI

Robin project, HACMS Contest week-end 14-15 June Summer School

7 Other works at SRI

Discovering PVS : Translating Coq proofs to PVS PVS library for basic linear algebra
HACMS with Robin Parsing Lisp code -¿ generate HTML architecture file Correcting translator PVS to SMT-LIB
Summer school
Try using bibtex here
– PVS –
PVS API Reference PVS Lisp sources (github rep) PVS language Reference PVS System Guide PVS Prelude library
– Lisp – Common Lisp Guy L. Steele Jr
– C– The C Library Reference Guide, Eric Huss
– Other – Compilation course, J.C. Filliatre
[2] [1] [3]

References

[1] S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Calvert. *User Guide for the PVS Specification and Verification System*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1998. Three volumes: Language, System, and Prover Reference Manuals.

[2] S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Calvert. *PVS Language Reference*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1999.

[3] S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Calvert. *PVS System Guide*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1999.