



## 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 août 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

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>PVS</b>	<b>2</b>
<b>3</b>	<b>Translating PVS to C</b>	<b>2</b>
<b>4</b>	<b>PVS Syntax</b>	<b>2</b>
<b>5</b>	<b>Types</b>	<b>3</b>
5.1	Fragment of PVS syntax . . . . .	4
5.2	Difficulties . . . . .	4
<b>6</b>	<b>Other activities at SRI</b>	<b>5</b>
<b>7</b>	<b>Other works at SRI</b>	<b>5</b>

# 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 [4].

```
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
```

<i>IfExpr</i>	::=	IF <i>Expr</i> THEN <i>Expr</i> { ELSIF <i>Expr</i> THEN <i>Expr</i> } * ELSE <i>Expr</i> ENDIF
<i>BindingExpr</i>	::=	<i>BindingOp</i> <i>LambdaBindings</i> : <i>Expr</i>
<i>BindingOp</i>	::=	LAMBDA   FORALL   EXISTS   { <i>IdOp</i> ! }
<i>LambdaBindings</i>	::=	<i>LambdaBinding</i> [ [, ] <i>LambdaBindings</i> ]
<i>LambdaBinding</i>	::=	<i>IdOp</i>   <i>Bindings</i>
<i>SetBindings</i>	::=	<i>SetBinding</i> [ [, ] <i>SetBindings</i> ]
<i>SetBinding</i>	::=	{ <i>IdOp</i> [ : <i>TypeExpr</i> ] }   <i>Bindings</i>
<i>Assignment</i>	::=	<i>AssignArgs</i> { :=    -> } <i>Expr</i>
<i>AssignArgs</i>	::=	<i>Id</i> [ ! <i>Number</i> ]   <i>Number</i>   <i>AssignArg</i> <sup>+</sup>
<i>AssignArg</i>	::=	( <i>Expr</i> <sup>+</sup> )   ' <i>Id</i>   ' <i>Number</i>
<i>Selection</i>	::=	<i>IdOp</i> [ ( <i>IdOps</i> ) ] : <i>Expr</i>
<i>TableExpr</i>	::=	TABLE [ <i>Expr</i> ] [ , <i>Expr</i> ] [ <i>ColHeading</i> ] <i>TableEntry</i> <sup>+</sup> ENDTABLE
<i>ColHeading</i>	::=	[ <i>Expr</i> {   { <i>Expr</i>   ELSE } } <sup>+</sup> ]
<i>TableEntry</i>	::=	{   [ <i>Expr</i>   ELSE ] } <sup>+</sup>
<i>LetBinding</i>	::=	{ <i>LetBind</i>   ( <i>LetBind</i> <sup>+</sup> ) } = <i>Expr</i>
<i>LetBind</i>	::=	<i>IdOp</i> <i>Bindings</i> * [ : <i>TypeExpr</i> ]
<i>Arguments</i>	::=	( <i>Expr</i> <sup>+</sup> )

## 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

<i>TypeExpr</i>	::=	<i>Name</i>   <i>EnumerationType</i>   <i>Subtype</i>   <i>TypeApplication</i>   <i>FunctionType</i>   <i>TupleType</i>   <i>CotupleType</i>   <i>RecordType</i>
<i>EnumerationType</i>	::=	{ <i>IdOps</i> }
<i>Subtype</i>	::=	{ <i>SetBindings</i>   <i>Expr</i> }   ( <i>Expr</i> )
<i>TypeApplication</i>	::=	<i>Name Arguments</i>
<i>FunctionType</i>	::=	[FUNCTION   ARRAY] [ -[ <i>IdOp</i> : ] <i>TypeExpr</i> <sup>+</sup> ; -> <i>TypeExpr</i> ]
<i>TupleType</i>	::=	[ -[ <i>IdOp</i> : ] <i>TypeExpr</i> <sup>+</sup> ]
<i>CotupleType</i>	::=	[ -[ <i>IdOp</i> : ] <i>TypeExpr</i> <sup>+</sup> <sub>+</sub> ]
<i>RecordType</i>	::=	[# <i>FieldDecls</i> <sup>+</sup> , #]
<i>FieldDecls</i>	::=	<i>Ids</i> : <i>TypeExpr</i>

type-expr $\subset$ <b>syntax</b>	[abstract class]
.....	
type-name $\subset$ <b>type-expr name</b> <i>adt</i>	[class]
.....	
subtype $\subset$ <b>type-expr</b> <i>supertype</i> <i>predicate</i>	[class]
.....	
funtype $\subset$ <b>type-expr</b> <i>domain</i> <i>range.</i>	[class]
.....	
tupletype $\subset$ <b>type-expr</b> <i>types</i>	[class]
.....	
recordtype $\subset$ <b>type-expr</b> <i>fields</i>	[class]
.....	

boolean, number, number\_field, real, rational, integer,  $A \rightarrow B$ , 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 -i 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.
- [4] N. Shankar and S. Owre. *PVS API Reference*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 2003.