

PVS Quick Reference

Reference to PVS Version 2.2

The commands which appear below are given with their abbreviations and keybindings, if any. For example, the command Prove with aliases `pr` and `C-c p` indicates that the parse command can be invoked by the Emacs extended commands `M-x prove` or `M-x pr`, or the key binding `C-c p`.

Entering and Exiting PVS

To enter PVS just `cd` to a working directory (*PVS context*) and type `pvs`.

<code>suspend-pvs</code>	<code>C-x C-z</code>
<code>exit-pvs</code>	<code>C-x C-c</code>

Getting Help

<code>help-pvs</code>	<code>C-c h</code>
<code>help-pvs-language</code>	<code>C-c C-h l</code>
<code>help-pvs-prover</code>	<code>C-c C-h p</code>
<code>help-pvs-prover-command</code>	<code>C-c C-h c</code>
<code>help-pvs-prover-strategy</code>	<code>C-c C-h s</code>
<code>help-pvs-prover-emacs</code>	<code>C-c C-h e</code>

Editing PVS Files

<code>forward-theory</code>	<code>M-}</code>
<code>backward-theory</code>	<code>M-{</code>
<code>find-unbalanced-pvs</code>	<code>C-c]</code>
<code>comment-region</code>	<code>C-c ;</code>

Parsing and Typechecking

<code>parse</code>	<code>M-x pa</code>
<code>typecheck</code>	<code>M-x tc, C-c C-t</code>
<code>typecheck-importchain</code>	<code>M-x tci</code>
<code>typecheck-prove</code>	<code>M-x tcp</code>
<code>typecheck-prove-importchain</code>	<code>M-x tcpi</code>

Typecheck Information

`show-theory-warnings`
`show-pvs-file-warnings`
`show-theory-messages`
`show-pvs-file-messages`

Prover Invocation Commands

<code>prove</code>	<code>M-x pr, C-c p</code>
<code>x-prove</code>	<code>M-x xpr, C-c C-p x</code>
<code>step-proof</code>	<code>M-x prs, C-c C-p s</code>
<code>x-step-proof</code>	<code>M-x xsp, C-c C-p X</code>
<code>redo-proof</code>	<code>M-x prr, C-c C-p r</code>
<code>prove-theory</code>	<code>M-x prt, C-c C-p t</code>
<code>prove-pvs-file</code>	<code>M-x prf, C-c C-p f</code>
<code>prove-importchain</code>	<code>M-x pri, C-c C-p i</code>
<code>prove-proofchain</code>	<code>M-x prp, C-c C-p p</code>

Proof Editing Commands

edit-proof, show-proof	
install-proof	C-c C-i
revert-proof	
remove-proof	
show-proof-file	
show-orphaned-proofs	
show-proofs-theory	
show-proofs-pvs-file	
show-proofs-importchain	
install-pvs-proof-file	
load-pvs-strategies	
set-print-depth	
set-print-length	
set-rewrite-depth	
set-rewrite-length	
toggle-proof-prettyprinting	

Proof Information Commands

show-current-proof
explain-tcc
show-last-proof
ancestry
siblings
show-hidden-formulas
show-auto-rewrites
show-expanded-sequent
show-skolem-constants

Adding and Modifying Declarations

add-declaration
modify-declaration

Prettyprinting Commands

prettyprint-theory	M-x ppt, C-c C-q t
prettyprint-pvs-file	M-x ppf, C-c C-q f
prettyprint-declaration	M-x ppd, C-c C-q d, C-M-q
prettyprint-region	M-x ppr, C-c C-q r, C-M-\

Viewing TCCs

prettyprint-expanded	M-x ppe, C-c C-q e
show-tccs	M-x tccs, C-c C-q s

PVS File and Theory Commands

find-pvs-file	M-x ff, C-c C-f
find-theory	M-x ft
view-prelude-file	M-x vpf
view-prelude-theory	M-x vpt
view-library-file	M-x vlf
view-library-theory	M-x vlt
new-pvs-file	M-x nf
new-theory	M-x nt
import-pvs-file	M-x imf
import-theory	M-x imt
delete-pvs-file	M-x df
delete-theory	M-x dt

save-pvs-buffer	C-x C-s
save-pvs-file	C-x C-s
save-some-pvs-files	M-x ssf
smail-pvs-files	
rmail-pvs-files	
dump-pvs-files	
undump-pvs-files	
edit-pvs-dump-file	

Printing Commands

pvs-print-buffer	
pvs-print-region	
print-theory	M-x ptt
print-pvs-file	M-x ptf
print-importchain	M-x pti
alltt-theory	M-x alt, C-c C-a t
alltt-pvs-file	M-x alf, C-c C-a f
alltt-importchain	M-x ali, C-c C-a i
alltt-proof	M-x alp, C-c C-a p
latex-theory	M-x ltt, C-c C-l t
latex-pvs-file	M-x ltf, C-c C-l f
latex-importchain	M-x lti, C-c C-l i
latex-proof	M-x ltp, C-c C-l p
latex-theory-view	M-x ltv, C-c C-l v
latex-proof-view	M-x lpv, C-c C-l P
latex-set-linlength	M-x lts, C-c C-l s

Display Commands

x-theory-hierarchy
x-show-proof
x-show-current-proof
x-prover-commands

Context Commands

list-pvs-files	M-x lf
list-theories	M-x lt
change-context	M-x cc
save-context	M-x sc
pvs-remove-bin-files	
pvs-dont-write-bin-files	
pvs-do-write-bin-files	
context-path	M-x cp

Library Commands

load-prelude-library
remove-prelude-library

Browsing Commands

show-declaration	M-.
find-declaration	M-,
whereis-declaration-used	M-;
whereis-identifier-used	C-M-;
list-declarations	M-:
goto-declaration	M-'

Status Commands

status-theory	M-x stt, C-c C-s t
status-pvs-file	M-x stf, C-c C-s f
status-importchain	M-x sti, C-c C-s i
status-importbychain	M-x stb, C-c C-s b
status-proof	M-x sp, C-c s p
status-proof-theory	M-x spt
status-proof-pvs-file	M-x spf
status-proof-importchain	M-x spi
status-proofchain	M-x spc
status-proofchain-theory	M-x spct
status-proofchain-pvs-file	M-x spcf
status-proofchain-importchain	M-x spci

Environment Commands

whereis-pvs	
pvs-version	
pvs-mode	
pvs-log	
status-display	
pvs-status	
remove-popup-buffer	C-z 1
pvs	
pvs-load-patches	
pvs-interrupt-subjob	C-c C-c
reset-pvs	C-z z

LaTeX Substitution Files

LaTeX Substitutions for file `foo.pvs` may come from any of the following files.

File name	Location
<code>foo.sub</code>	the directory of the current context
<code>pvs-tex.sub</code>	the directory of the current context
<code>pvs-tex.sub</code>	user's home directory
<code>pvs-tex.sub</code>	the main PVS directory

Examples of substitution entries—numbers refer to the number of arguments; thus the third entry translates `f2[3,G]` (to G_3^f) but not `f2[int]`, and the last entry translates, e.g., `f4(G)(1,n)` (to $\sum_{i=1}^n G(i,1)$). Length is an estimation of the size of the translation, ignoring the size of the arguments.

Identifier	Type	Length	Substitution
THEORY	key	9	<code>{\large\bf Theory}</code>
f1	id	3	<code>{\rm bar}</code>
f2	id[2]	2	<code>{#2_{#1}^{\f}}</code>
f3	2	2	<code>{#1^{\#2}}</code>
f4	(1 2)	3	<code>{\sum_{i=#2}^{\#3} #1(i,#2)}</code>

PVS Files

<code>foo.pvs</code>	Specification file (contains theories)
<code>foo.bin</code>	Binary form of the typed specification file
<code>foo.prf</code>	Saved proofs for <code>foo.pvs</code>
<code>.pvscontext</code>	Context information
<code>foo-alltt.tex</code>	Alltt-printed version of <code>foo</code>
<code>foo.tex</code>	LaTeX-printed version of <code>foo</code>
<code>pvs-files.tex</code>	LaTeX file generated for testing Alltt and LaTeX-printed files

PVS Prover Commands

Prover commands are entered in the `*pvs*` buffer at the Rule? prompt. Commands are interpreted by Lisp, so must be surrounded by parentheses, and arguments are separated by whitespace (Space, Tab, or Return). Some commands require PVS names or expressions; these must be surrounded by double quotes ("). Return enters the command, unless parentheses or strings are unbalanced. The arguments are shown in *emphasized* font. Optional arguments follow the keyword `&OPTIONAL` and may be omitted. They may be provided in the order listed, or followed by a keyword whose name is derived from the argument name preceded by a colon, *e.g.*, `(expand "foo" :beta-reduce t)`. An `&rest` keyword indicates that one or more of the following argument may be provided, and may also be given in keyword form.

Help

(help &OPTIONAL *name[*]*)

Control

(fail)
(postpone)
(quit)
(rewrite-msg-off)
(rewrite-msg-on)
(skip)
(skip-msg *string* &OPTIONAL *force-printing?*)
(undo &OPTIONAL *to[1]*)

Structural Rules

(copy *fnum*)
(delete &rest *fnums*)
(hide &rest *fnums*)
(reveal &rest *fnums*)

Propositional Rules

(bddsimp &OPTIONAL *fnums[*]* *dynamic-ordering?*)
(case &rest *exprs*)
(case* &rest *exprs*)
(flatten &rest *fnums[*]*)
(iff &rest *fnums[*]*)
(lift-if &rest *fnums* *updates?[t]*)
(prop)
(propax)
(split &OPTIONAL *fnum[*]*)
(merge-fnums *fnums*)

Quantifier Rules

(detuple-boundvars &OPTIONAL *fnums[*]* *singles?*)
(generalize *term var* &OPTIONAL *type fnums[*]* *subterms-only?[t]*)
(generalize-skolem-constants &OPTIONAL *fnums[*]*)
(inst *fnum* &rest *terms*)
(instantiate *fnum terms* &OPTIONAL *copy?*)
(inst-cp *fnum* &rest *terms*)
(inst? &OPTIONAL *fnums[*]* *subst where[*]* *copy? if-match polarity?*)
(skolem *fnum constants*)
(skolem! &OPTIONAL *fnum[*]* *keep-underscore?*)
(skolem-typepred &OPTIONAL *fnum[*]*)
(skosimp &OPTIONAL *fnum[*]* *preds?*)

(skosimp* &OPTIONAL *preds?*)

Equality Rules

(beta &OPTIONAL *fnums[*] rewrite-flag*)
(case-replace *expr*)
(name *name expr*)
(name-replace *name expr* &OPTIONAL *hide?[T]*)
(name-replace* *name-and-exprs* &OPTIONAL *hide?[T]*)
(replace *fnum* &OPTIONAL *fnums[*] dir[LR] hide? actuals?*)
(replace* *&rest fnums*)
(same-name *name1 name2* &OPTIONAL *type*)

Definition and Lemma Rules

(expand *name* &OPTIONAL *fnum[*] occurrence if-simplifies assert?*)
(expand* *&rest names*)
(forward-chain *name-or-fnum*)
(lemma *name* &OPTIONAL *subst*)
(rewrite *name* &OPTIONAL *fnums[*] subst target-fnums[*] dir[LR] order[IN]*)
(rewrite-lemma *lemma subst* &OPTIONAL *fnums[*] dir[LR]*)
(rewrite-with-fnum *fnum* &OPTIONAL *subst fnums[*] dir[LR]*)
(use *lemma* &OPTIONAL *subst if-match[best]*)
(use* *&rest names*)

Extensionality Rules

(apply-eta *term* &OPTIONAL *type*)
(apply-extensionality &OPTIONAL *fnum[+] keep? hide?*)
(decompose-equality &OPTIONAL *fnum[*] hide?[t]*)
(eta *type*)
(extensionality *type*)
(replace-eta *term* &OPTIONAL *type keep?*)
(replace-extensionality *expr1 expr2* &OPTIONAL *expected keep?*)

Induction Rules

(induct *var* &OPTIONAL *fnum[1] name*)
(induct-and-rewrite *var* &OPTIONAL *fnum[1] &rest rewrites*)
(induct-and-rewrite! *var* &OPTIONAL *fnum[1] &rest rewrites*)
(induct-and-simplify *var* &OPTIONAL *fnum[1] name defs[T] if-match[best] theories rewrites exclude*)
(measure-induct *measure vars* &OPTIONAL *fnum[1] order*)
(measure-induct+ *measure vars* &OPTIONAL *fnum[1] order*)
(measure-induct-and-simplify *measure vars* &OPTIONAL *fnum[1] order expand defs[T] if-match[best] theories rewrites exclude*)
(name-induct-and-rewrite *var* &OPTIONAL *fnum[1] name &rest rewrites*)
(rule-induct *rel* &OPTIONAL *fnum[+] name*)

Decision Procedure and Rewriting Rules

(assert &OPTIONAL *fnums[*] rewrite-flag flush? linear? cases-rewrite? type-constraints?[t]*)
(bash &OPTIONAL *if-match[T] updates?[T] polarity?*)
(both-sides *op term* &OPTIONAL *fnum[1]*)
(do-rewrite &OPTIONAL *fnums[*] rewrite-flag flush? linear? cases-rewrite? type-constraints?[t]*)
(grind &OPTIONAL *defs[!] theories rewrites exclude if-match[T] updates?[T] polarity?*)

```

(ground)
(record &OPTIONAL fnums[*] rewrite-flag flush? linear? cases-rewrite?
  type-constraints[t])
(reduce &OPTIONAL if-match[T] updates?[T] polarity?)
(simplify &OPTIONAL fnums[*] record? rewrite? rewrite-flag
  flush? linear? cases-rewrite? type-constraints?[t])
(smash &OPTIONAL updates?[T])

```

----- Installation of Rewrite Rules -----

```

(auto-rewrite &rest names)
(auto-rewrite! &rest names)
(auto-rewrite!! &rest names)
(auto-rewrite-defs &OPTIONAL explicit? always? exclude-theories)
(auto-rewrite-explicit &OPTIONAL always?)
(auto-rewrite-theories &rest theories)
(auto-rewrite-theory &rest names)
(install-rewrites &OPTIONAL defs theories rewrites exclude-theories
  exclude)

```

----- Removing Installed Rewrite Rules -----

```

(stop-rewrite &rest names)
(stop-rewrite-theory &rest names)

```

----- Tracking Rewrite Rules -----

```

(trace &rest names)
(track-rewrite &rest names)
(untrace &rest names)
(untrack-rewrite &rest names)

```

----- Miscellaneous Rules -----

```

(apply strategy &OPTIONAL comment)
(model-check &OPTIONAL dynamic-ordering?[T] cases-rewrite?[T])
(musimp &OPTIONAL fnums[*] dynamic-ordering?[nil])
(typepred &rest exprs)
(typepred! exprs &OPTIONAL all?)

```

----- Strategy Constructors -----

```

(branch step steplist)
(else step1 step2)
(if condition step1 step2)
(let ((var1 expr1)... (varn exprn)) step)
(query*)
(quote step)
(repeat step)
(repeat* step)
(rerun &OPTIONAL proof)
(spread step steplist)
(spread! step steplist)
(spread@ step steplist)
(then &rest steps)
(then@ &rest steps)
(time step)
(try step1 step2 step3)
(try-branch step1 steplist step2)

```

Prover Emacs Shortcuts

These commands are only available when a proof is in progress, and the `*pvs*` buffer is current.

Prover Emacs Help	TAB h
Prover Command Help	TAB H
Any Command	TAB TAB
apply-extensionality	TAB E
assert	TAB a
auto-rewrite	TAB A
auto-rewrite-theory	TAB C-a
bddsimp	TAB B
beta	TAB b
case	TAB c
case-replace	TAB C
copy	TAB 2
decompose-equality	TAB =
delete	TAB d
do-rewrite	TAB D
expand	TAB e
extensionality	TAB x
flatten	TAB f
grind	TAB G
ground	TAB g
hide	TAB C-h
iff	TAB F
induct	TAB I
induct-and-simplify	TAB C-s
inst	TAB i
inst?	TAB ?
lemma	TAB L
lift-if	TAB l
model-check	TAB M
musimp	TAB m
name	TAB n
name-replace	TAB N
postpone	TAB P
prop	TAB p
quit	TAB C-q
replace	TAB r
replace-eta	TAB 8
rewrite	TAB R
skolem!	TAB !
skosimp	TAB S
skosimp*	TAB *
split	TAB s
tcc	TAB T
then	TAB C-t
typepred	TAB t
undo	TAB u
One Proof Step	TAB 1
Many Proof Steps	TAB @
Undo One Proof Step	TAB U
Undo Many Proof Steps	TAB C-u
Skip Proof Step	TAB #
Insert Quotes	TAB '
Wrap with Parends	TAB C-j

PVS Language Examples

Theories

```
function_properties [D, R: TYPE]: THEORY
BEGIN
  f, g: VAR [D -> R]
  x, x1, x2: VAR D
  y: VAR R
  injective?(f): bool =
    (FORALL x1, x2: (f(x1) = f(x2) => (x1 = x2)))
  surjective?(f): bool =
    (FORALL y: (EXISTS x: f(x) = y))
END function_properties

finite[t: TYPE]: THEORY
BEGIN
  IMPORTING function_properties
  is_finite_type: bool =
    (EXISTS (n:nat), (f:[upto[n] -> t]): surjective?(f))
  is_finite_type_alt: LEMMA
    is_finite_type IFF
      (EXISTS (n:nat), (g:[t -> upto[n]]): injective?(g))
END finite

best_choice[t: TYPE, meas: TYPE FROM real]: THEORY
BEGIN
  ASSUMING
    IMPORTING finite[t]
    finite: ASSUMPTION is_finite_type[t]
  ENDASSUMING
  best: [[t -> meas], setof[t] -> t]
  f: VAR [t -> meas]
  s: VAR setof[t]
  best_ax: AXIOM
    nonempty?(s) => member(best(f, s), s)
    AND (FORALL (x: t): member(x, s) => f(x) <= f(best(f, s)))
END best_choice
```

Lexical Rules

Comments start with % and go to the end of the line

Identifiers are composed of letters, digits, question mark, and underscores; they must begin with a letter and are case-sensitive.

Numbers are composed of digits—no floating point numbers.

Strings are enclosed in double quotes "astring"

Reserved Words

Reserved words are not case sensitive.

ALL	CONJECTURE	FACT	LAW	SUBLEMMA
AND	CONTAINING	FALSE	LEMMA	SUBTYPE_OF
ANDTHEN	CONVERSION	FORALL	LET	SUBTYPES
ARRAY	COROLLARY	FORMULA	LIBRARY	TABLE
ASSUMING	DATATYPE	FROM	MEASURE	THEN
ASSUMPTION	ELSE	FUNCTION	NONEMPTY_TYPE	THEOREM
AXIOM	ELSIF	HAS_TYPE	NOT	THEORY
BEGIN	END	IF	O	TRUE
BUT	ENDASSUMING	IFF	OBLIGATION	TYPE
BY	ENDCASES	IMPLIES	OF	TYPE+
CASES	ENDCOND	IMPORTING	OR	VAR
CHALLENGE	ENDIF	IN	ORELSE	WHEN
CLAIM	ENDTABLE	INDUCTIVE	POSTULATE	WHERE
CLOSURE	EXISTS	JUDGEMENT	PROPOSITION	WITH
COND	EXPORTING	LAMBDA	RECURSIVE	XOR

Special Symbols

&	()	*	+	,
-	->	.	/	/=	:
:=	;	<	<=	<=>	=
=>	>	>=		[]
[#	#]	{	}	(#	#)
\	^	~	==	□	<>
^	@		!]	->
[]	(:	:)	!	[]
@@	##	**	++	//	^^
-	=	<	>	<<	>>
<<=	>>=	#			

Precedence Table

Operators	Associativity
FORALL, EXISTS, LAMBDA, IN	None
	Left
-, =	Right
IFF, <=>	Right
IMPLIES, =>, WHEN	Right
OR, \/, XOR, ORELSE	Right
AND, &, &&, /\, ANDTHEN	Right
NOT, ~	None
=, /=, ==, <, <=, >, >=, <<, >>, <<=, >>=, < , >	Left
WITH	Left
WHERE	Left
@, #	Left
@@, ##,	Left
+, -, ++,	Left
, /, **, //	Left
-	None
o	Left
:, ::, HAS_TYPE	Left
□, <>	None
~, ^^	Left

Type Declarations

- Uninterpreted types
 - `foo`: TYPE
 - `bar`: NONEMPTY_TYPE % same as TYPE+
 - `some_nums`: NONEMPTY_TYPE FROM number
- Subtypes
 - `nat_to_10`: TYPE = {x:nat | x <= 10}
 - `posint`: TYPE = {x:integer | x > 0} CONTAINING 1
 - `ptype`: TYPE = (pred?) % same as {x | pred?(x)}
 - `rtype`: TYPE = {x, y: nat | x < y} % subtype of [nat, nat]
- Function types
 - `intf`: TYPE = FUNCTION[int, int -> int]
 - `altf`: TYPE = [int, int -> int] % same as above
 - `inta`: TYPE = ARRAY[int,int -> int] % same as above
- Tuple Types
 - `tuptype`: TYPE = [int, bool, [int -> int]]
- Record types
 - `stack`: TYPE = [# pointer: nat,
 astack: [nat -> t] #]
- Dependent Types
 - `pfun`: TYPE = [# dom: predicate[t1], pfn:[(dom)->t2] #]
 - `date`: TYPE = [y,m:nat, {d:nat | d <= days(m,y)}]
 - `tmod`: TYPE = [n,m:int -> {x:nat | x < m}]
- Enumeration types
 - `color`: TYPE = {red, green, blue}
- Datatypes
 - `list[t:TYPE] : DATATYPE`
BEGIN
 `null`: null?
 `cons` (car: t, cdr :list) :cons?
END list
 - `expression`: DATATYPE WITH SUBTYPES term, typ
BEGIN
 `base_type`(n:nat): base_type? : typ
 `funtype`(dom: typ, ran: typ): funtype? : typ
 `variable`(n:nat): variable? : term
 `number`(num:nat): number? : term
 `lam`(v: (variable?), ty: typ, ex: term): lam? : term
 `app`(op: term, arg: term): app? : term
END expression

Libraries, Importings, Exportings, and Theory Abbreviations

- `fsets: LIBRARY = "/homes/pvs/lib/finite_sets"`
- `IMPORTING orderings[int], set[foo[nat]],
 fsets@finite_sets[nat]`
- `EXPORTING foo, bar WITH set[foo]`
- `pset: THEORY = sets[list[nat]]`

Constants and Recursive Definitions

- `some_int: int`
- `max: int = 10`
- `abs: [int -> nat] =
 (LAMBDA x: IF x < 0 THEN -x ELSE x ENDIF)`
- `abs(x:int): nat = IF x < 0 THEN -x ELSE x ENDIF`
- `sum(f,x,y): int % f,x,y prev declared VAR`
- `sum(f,(x,y:int)): int % f prev declared VAR`
- `fac(n): RECURSIVE nat =
 (IF n = 0 THEN 1 ELSE n*fac(n-1) ENDIF)
 MEASURE (LAMBDA n: n)`
- `length(l:list): RECURSIVE nat =
 CASES 1 OF
 null: 0,
 cons(x, y): length(y) + 1
 ENDCASES
 MEASURE 1 BY << % Subterm measure`

Variable Declarations

- `x, y, z: VAR int`
- `f: VAR [int -> [int -> int]]`

Formula Declarations

- `transitive: AXIOM x < y AND y < z IMPLIES x < z`
- `nonzero_fac: THEOREM fac(n) /= 0`
- `poset: ASSUMPTION poset?(T,<=) % Only in ASSUMINGS`

Judgements

- `JUDGEMENT {x :int | x > 10} SUBTYPE_OF posint`
- `JUDGEMENT c HAS_TYPE (even?)`
- `JUDGEMENT +, -, * HAS_TYPE [(even?), (even?) -> (even?)]`

Conversions

- `C: [int -> bool] = (LAMBDA (i:int): i=0)
 CONVERSION C
 foo: FORMULA d + 1 % \equiv foo: FORMULA C(d + 1)`
- `state: TYPE
K: [int -> [state -> int]] = (LAMBDA i: (LAMBDA s: i))
f: [[state -> int] -> [state -> int]]
x: [state -> int]
B: [[state -> int] -> bool] = (LAMBDA si: FORALL i: si(i))
 CONVERSION K, B
 bar: LEMMA f(x+1) % \equiv bar: LEMMA B(f(LAMBDA s: x(s)+1))`

Expressions

- Equality — (=, /=)
Defined for any type; both sides must be the same type.
With boolean, = is treated as IFF.
 - `x * y = 4`
 - `true /= 1` % Illegal
- Arithmetic — (+, -, *, /, <, <=, >, >=, 0, 1, ...)
 - `((x + 1) * x) / 2 < x * x`
- Lists and Strings
 - `(: 1, 2 :)` % \equiv `cons(1, cons(2, null))`
 - `"A string"` % A finite sequence of characters
- Logical — (true, false, AND, &, OR, IMPLIES, =>, WHEN, NOT, IFF, <=>, FORALL, ALL, EXISTS, SOME)
 - `(FORALL e: (EXISTS d: abs(f(x) - f(y)) < d) IMPLIES abs(x - y) < e)`
- IF-THEN-ELSE — The THEN and ELSE parts must have compatible types.
 - `IF x = 0 THEN 1 ELSIF y = 0 THEN 2 ELSE y/x ENDIF`
- CASES — Pattern matching on datatypes.
 - `CASES x OF
 cons(x,y): append(reverse(y), cons(x, null))
 ELSE null
ENDCASES`
- COND — generates coverage and disjointness TCCs
 - `COND m = n -> m,
 m > n -> gcd(m-n, n),
 m < n -> gcd(m, n-m)
ENDCOND`
 - `COND m = n -> m,
 m > n -> gcd(m-n, n),
 ELSE -> gcd(m, n-m)
ENDCOND % Same as above, but no coverage TCC`
- Function application, lambda-abstraction & function update
 - `f(1,2)(0)`
 - `(lambda x: x + 1)`
 - `f WITH [(0) := 1, (1) := 0]`
 - `foo ! (x:int): e` % \equiv `foo(LAMBDA (x: int): e)`
- Set expressions
 - `{x: int | x < 10}` % same as `(LAMBDA (x: int): x < 10)`
- Record construction, field selection & record update
 - `(# pointer := 1, astack := (LAMBDA x: 0) #)`
 - `astack(r)` % `r.astack` NOT allowed
 - `r WITH [pointer := 2, (astack)(1) := 1]`

- Tuple construction, projection, and update
 - `(1, true, (LAMBDA (x:int) x + 37))`
 - `proj_3(tup)`
 - `tup WITH [2 := false]`

- LET & WHERE
 - `LET x = 2, y:nat = x*x IN f(x,y) % ≡ f(2,4)`
 - `f(x,y) WHERE x = 2, y:nat = x*x % same`
 - `LET (x, y, z) = t IN x + y * z % same as next line`
 - `LET x=PROJ_1(t), y=PROJ_2(t), z=PROJ_3(t) IN x + y * z`

- Coercion — Coercion indicates the expected type to the type-checker to resolve ambiguity.
 - `a + b:natural`
 - `(LAMBDA n -> nat: n - m) % LAMBDA coercion`

- Names — If `foo` is declared in theory `bar`, then the following are allowable references (the first two may be ambiguous).
 - `foo`
 - `foo[int]`
 - `bar[int].foo`