Theory Interpretations

Consistency Relative to PVS

César A. Muñoz

NASA Langley Research Center
Cesar.A.Munoz@nasa.gov

Logic 101

- A *(formal) system* is inconsistent if we can prove both A and $\neg A$.
- A consistency proof is hard. It is easier to prove that a system is consistent relative to another system (believed to be consistent itself).
- Relative consistency is shown by exhibiting model.

Inconsistency, It Can Happen to You

- This axiom is found in a highly referenced paper written in 1999:

```
C    : [real-> int]
MyAx : AXIOM
  FORALL(x,y:real): x < y IMPLIES C(x) < C(y)
```

- The theory is inconsistent as `MyAx` implies that real numbers are enumerable.

- The inconsistency was revealed by L. Pike using *theory interpretations*.

Theory Interpretations in PVS

- A mechanism to construct models of axiomatic PVS theories, by instantiating uninterpreted types and constants.
- Consistency relative to the PVS logic can be shown via theory interpretations.

Is This Theory Consistent?

(Relative to the PVS System)

```
myTh : THEORY
BEGIN
  C    : [real-> int]
  x,y  : VAR real

  MyAx : AXIOM
    x < y IMPLIES C(x) <= C(y)
END myTh
```

For the Impatient

```
myTh : THEORY
BEGIN
  C    : [real-> int]
  x,y  : VAR real

  MyAx : AXIOM
    x < y IMPLIES
    C(x) <= C(y)
END myTh
```

```
myThi : THEORY
BEGIN
  IMPORTING myTh{{
     C(x:real) := floor(x)
  }}
END myThi

IMP_myTh_MyAx_TCC1: OBLIGATION
  FORALL (x, y: real): x < y
  IMPLIES floor(x) <= floor(y);
```

Isn't that What Theory Parameters are for?

```
myThp[C:[real->int]] : THEORY
BEGIN
  MyAx : AXIOM
    FORALL(x,y:real): x < y
    IMPLIES C(x) <= C(y)
END myThp

myThpi : THEORY
BEGIN
  IMPORTING myThp[floor]
END myThpi
```

No. Theory parameters do not generate proof obligations for the axioms in the source theory.

Can This Be Done With Subtying, Assumptions, . . . ?

```
myThp1[C:[real->int]] : THEORY
BEGIN
  ASSUMING
    MyAx : ASSUMPTION
      FORALL(x,y:real): x < y
      IMPLIES C(x) <= C(y)
  ENDASSUMING
END myThp1

myThp2[C:c:[real->int]| FORALL(x,y:real):
       x < y IMPLIES c(x) <= c(y)] : THEORY
BEGIN
END myThp2
```

Possibly.

Theory Parameters vs. Theory Interpretation

Theory interpretations largely subsume theory parameters, but

- Theory parameters are intended for the specification of a family of problems.
- Theory interpretations are intended for
  - Checking the consistency of an axiomatic specification.
  - Reification of an abstract data type.
  - Animation of an axiomatic specification.

Outline

Consistency Checking

```
th : THEORY
BEGIN
  T : TYPE+
  i : T
  o : [[T,T]->T]
  x,y,z  : VAR T

  id    : AXIOM x o i = x
  assoc : AXIOM (x o y) o z = x o (y o z)
  inv   : AXIOM EXISTS(y): x o y = i AND y o x = i

  di : LEMMA
    i o x = x
END th
```

Return ThAsParam Return ThAsDecl

A Model or Two (Via Theory Abbreviations)

```
thi : THEORY
BEGIN

  IMPORTING th{{ T:=real,   i:=0, o(a,b:real  ):=a+b }}
              AS th0,
            th{{ T:=nzreal, i:=1, o(a,b:nzreal):=a*b }}
              AS th1

END thi
```

Proof Obligations as TCCs

```
h0_id_TCC1: OBLIGATION FORALL (x:real): x+0 = x;

th0_assoc_TCC1: OBLIGATION FORALL (x,y,z:real):
  x+y+z = x+(y+z);

th0_inv_TCC1: OBLIGATION FORALL (x:real):
  EXISTS (y:real): x+y = 0 AND y+x = 0;

th1_id_TCC1: OBLIGATION FORALL (x:nzreal): x*1 = x;

th1_assoc_TCC1: OBLIGATION FORALL (x,y,z:nzreal):
  x*y*z = x*(y*z);

th1_inv_TCC1: OBLIGATION FORALL (x:nzreal):
  EXISTS (y:nzreal): x*y = 1 AND y*x = 1;
```

To Be or Not to Be Consistent

- Claim: The theory `th` is consistent if TCCs in `thi` can be discharged.
- Remark: The above claim is made at the level of the PVS meta-theory, i.e., it is an external observation rather than a fact formally specified/proven in PVS.
- Note: No TCCs will be generated for an axiom

  ```
  foo : AXIOM 1=0
  ```

  in `th`.
- Question: Why ?

Reification of Abstract Data Types

Process of making a concrete type from an abstract data type.

Reminder:

- Abstract data types in PVS, i.e., `DATATYPE`s, are axiomatically defined.
- Enumeration types are abstract data types.

Enumeration Types are Abstract Data Types

```
states : THEORY
BEGIN
  State : TYPE = {idle,waiting,running}
END states

states_as_nat : THEORY
BEGIN
  NatState : TYPE = below[3]
  n        : VAR NatState
  IMPORTING states{{ State := NatState,
    idle?(n) := n=0, waiting?(n) := n=1, running?(n) := n=2,
    idle := 0, waiting := 1, running := 2}}
END states_as_nat
```

Proof Obligations

```
IMP_states_State_inclusive_TCC1: OBLIGATION
  FORALL (State_var:NatState):
    State_var = 0 OR State_var = 1 OR State_var = 2;

IMP_states_State_induction_TCC1: OBLIGATION
  FORALL (p:[NatState -> boolean]):
    p(0) AND p(1) AND p(2) IMPLIES
    (FORALL (State_var: NatState): p(State_var));
```

Note that `IMP_states_State_induction_TCC1` becomes unprovable if `NatState = nat`.

Animation of Specifications

Animation is the execution of a specification to validate its intended semantics.

- Animations in PVS are mostly performed in the Ground Evaluator.
- PVSio is a PVS package that re-implements the interface to the Ground Evaluator: `http://shemesh.larc.nasa.gov/people/cam/PVSio`.
- Wait for the talk on PVSio.

Lost in Translation?

- The Emacs command `M-x ppti` displays in a new buffer the result of a theory interpretation.
- Technical Report: *Theory Interpretations in PVS*, S. Owre and N. Shankar, SRI-CSL-01-01. Available from `http://pvs.csl.sri.com/documentation.shtml`.
- PVS Release notes available from http://pvs.csl.sri.com/download.shtml.

Reification of ADTs

```
list [T: TYPE]: DATATYPE
BEGIN
  null: null?
  cons (car: T, cdr:list):cons?
END list
```

Lists as Arrays

```
list_as_array[T:TYPE] : THEORY
BEGIN

  List : TYPE = [#
    length : nat,
    elems  : [below(length)->T]
  #]

  l : VAR List
  t : VAR T
```

Lists Constructors

```
Null?(l):bool = length(l) = 0

Null : List = (#
  length := 0,
  elems  := LAMBDA(x:below(0)):epsilon(emptyset[T])
#)

Cons?(l):bool = not Null?(l)

Cons(t,l): List = l WITH [
  `length          := l`length+1,
  `elems(l`length) := t
]
```

Interpretation

```
    IMPORTING list[T]{{ list  := List,
                        null? := Null?,
                        cons? := Cons?,
                        null  := Null,
                        cons  := Cons }}
  END list_as_arrays
```

PVS Lists are Consistent

```
IMP_list_TCC1: OBLIGATION
  Null?(Null);

IMP_list_TCC2: OBLIGATION
  FORALL (x1: [T, List]):
    Cons?(Cons(x1));

IMP_list_list_null_extensionality_TCC1: OBLIGATION
  FORALL (null?_var:  {x: List | Null?(x)},
          null?_var2: {x: List | Null?(x)}):
    null?_var = null?_var2;
```

Note that the record type where `elems : [nat->T]` does not directly yield a model of `list[T]`.*

*In that case, the model has to be constructed using quotient types.

Theories as Parameters

Assume that we want to extend the theory `th` with a definition for the inverse function:

```
inverse(x:T):{y:T | x o y = i}
```

Extending an Axiomatic Theory (The Wrong Way)

```
thx2 : THEORY
BEGIN
  IMPORTING th

  inverse(x:T):{y:T | x o y = i}
END thx2
```

Theory `thx2` does not provide a mechanism to construct an interpretation of `th`.

Theories as Parameters

```
thx [t:THEORY th] : THEORY
BEGIN
  inverse(x:T):{y:T | x o y = i}
END thx
```

An Interpretation of thx

```
thxi : THEORY
BEGIN
  IMPORTING thx[th{{T:=nzreal,i:=1,o(a,b:nzreal):=a*b}}]

  inv_def : LEMMA
    FORALL(a:nzreal): inverse(a) = 1/a
END thxi
```

Theory Declarations

Assume that we want define a theory like `th` but with an extra commutativity axiom:

```
commutativity : AXIOM
   FORALL(x,y:T): x o y = y o x
```

Extending an Axiomatic Theory (The Wrong Way)

```
thax2[t: THEORY th] : THEORY
BEGIN
  commutativity : AXIOM
    FORALL(x,y:T): x o y = y o x
END thax2

thaxi2 : THEORY
BEGIN
 IMPORTING
  thax2[ th {{ T:=nzreal,i:=1,o(a,b:nzreal):=a*b}} ]
END thaxi2
```

Theory `thaxi2` does not generate a TCC for the commutativity axiom.

Theory Declarations

```
thax : THEORY
BEGIN
   t : THEORY = th

   commutativity : AXIOM
     FORALL(x,y:T): x o y = y o x
END thax
```

An Interpretation of thax

```
thaxi : THEORY
BEGIN
  IMPORTING
  thax{{t := th {{ T:=nzreal,i:=1,o(a,b:nzreal):=a*b}} }}
END thaxi
```

More Theory Declarations

- `t1 : THEORY = th {{ T := nzreal }}`
  `t1` is a copy of `th` where `T` is substituted by `nzreal`. All the rest is left uninterpreted. Axioms related to `T` are generated as `t1`'s TCCs.
- `t3 : THEORY = th {{ T ::= myT }}`
  `t1` is a copy of `th` where `T` is renamed `myT`, which is uninterpreted. No TCCs are generated for `t3`.

Same-name Interpretations

The notation

```
IMPORTING th :-> thi
```

is syntactic sugar for

```
IMPORTING th{{ x_1 := thi.x_1, ..., x_n := thi.x_n }}
```

where `x_1`,...,`x_n` are identifiers with the same name in `th` and `thi`.