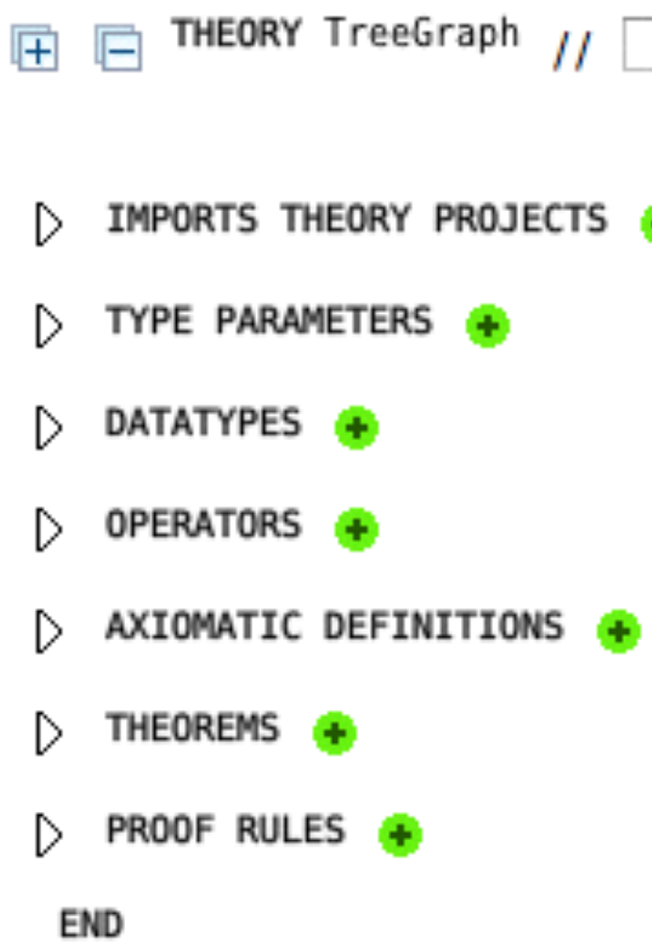
A Practical Approach for Validation with Rodin Theories

Daniel Plagge
Michael Leuschel

Rodin Theories & New Operators



Operators:

1. Direct Definition
2. Recursive Definition
3. Axiomatic Definition

Goal: validate models using theories

1. Direct Definitions

- “Simply” replace operator use by definition
- Often infinite lambdas or set comprehensions used
- Well-definedness issues discovered (if-then-else)

```
THEORY
  if_then_else
TYPE PARAMETERS
  A
OPERATORS
  •if  :  if(test : BOOL, then : A, else : A)  EXPRESSION  PREFIX
  direct definition
   if(test : BOOL, then : A, else : A) ≙ ((λx·x∈BOOL ∧ (test=TRUE)|then) ∪ (λx·x∈BOOL ∧ (test=FALSE)|else))(TRUE)
THEOREMS
  thm1   :   ∀x,y·x∈A ∧ y∈A ⇒ if(TRUE,x,y)=x
  thm2   :   ∀x,y·x∈A ∧ y∈A ⇒ if(FALSE,x,y)=y
  thm3   :   ∀x,b·x∈A ∧ b∈BOOL ⇒ if(b,x,x)=x
END
```

Infinite Direct Definitions

- Some Theory operators are infinite functions
  - seq(a) = {f,n · n ∈ ℕ ∧ f ∈ 1..n⟶a|f}
- ProB needs to deal with them
  - “automatic” detection of potentially infinite functions / set comprehensions
  - seq(0..1) ∩ { ∅, {1↦0}, {1↦2}} = { ∅, {1↦0}}
  - ProB keeps track of “enumeration warnings”, if it occurs during expansions we keep set comprehension symbolic (cf. Disprover/Prover talk tomorrow SETS workshop)

Seq

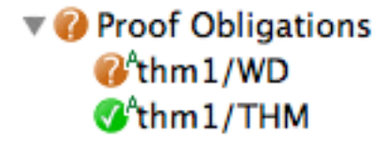
- Works out of the box

```
THEORY
          //   A theory of sequences defined as finite partial functions.
  Seq     //   @author Issam Maamria
          //   @date 17/07/2011
TYPE PARAMETERS
  A
OPERATORS
  •seq  :  seq(a : ℙ(A))  EXPRESSION  PREFIX        //   The sequence operator
  direct definition
   seq(a : ℙ(A)) ≙ {f,n · n ∈ ℕ ∧ f ∈ 1‥      //   a set of finite total function
   n⟶a|f}
  •seqSize  :  seqSize(s : seq(A))  EXPRESSION  PREFIX        //   size of seq
  direct definition
   seqSize(s : seq(A)) ≙ card(s)
  •seqIsEmpty  :  seqIsEmpty(s : seq(A))  PREDICATE  PREFIX        //   predic
                                                                   //   whethe
  direct definition
   seqIsEmpty(s : seq(A)) ≙ seqSize(s)=0
  •seqHead  :  seqHead(s : seq(A))  EXPRESSION  PREFIX        //   the head of
                                                              empty sequence
  well-definedness condition
   ¬ seqIsEmpty(s)
  direct definition
   seqHead(s : seq(A)) ≙ s(1)
  •seqTail  :  seqTail(s : seq(A))  EXPRESSION  PREFIX        //   the tail of
                                                              empty sequence
  well-definedness condition
   ¬ seqIsEmpty(s)
  direct definition
   seqTail(s : seq(A)) ≙ λi·i∈1‥seqSize(s)−1|s(i+1)
  •seqPrepend  :  seqPrepend(s : seq(A), e : A)  EXPRESSION  PREFIX        //
  direct definition
   seqPrepend(s : seq(A), e : A) ≙ {1↦e}∪(λi·i∈2‥seqSize(s)+1|s(i−1))
  •seqAppend  :  seqAppend(s : seq(A), e : A)  EXPRESSION  PREFIX        //
  direct definition
   seqAppend(s : seq(A), e : A) ≙ s∪{(seqSize(s)+1)↦e}
  •seqConcat  :  (s1 : seq(A)) seqConcat (s2 : seq(A))    EXPRESSION  INFIX
  direct definition
   (s1 : seq(A)) seqConcat (s2 : seq(A))  ≙ s1 ∪ (λi·i∈seqSize(s1)+1‥
   seqSize(s1)+seqSize(s2)|s2(i−seqSize(s1)))
```

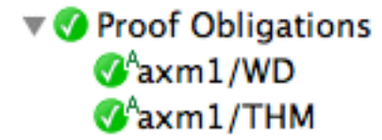
WD-Issues

- if(**test**,**then**,**else**) = ((λx·x∈BOOL ∧ (**test**=TRUE)⁞**then**) ∪
  (λx·x∈BOOL ∧ (**test**=FALSE)⁞**else**))(TRUE)

- ∀x·x∈0..10 ⇒ if(bool(x=0),1,**10÷x**)≥1



- if(bool(0=0),1,**10÷0**)≥1



- ((λx·x∈BOOL ∧ (bool(0=0)=TRUE)⁞1)
  ∪ (λx·x∈BOOL ∧ (bool(0=0)=FALSE)⁞10÷0))
  (TRUE) ≥ 1

- Theory Plugin's WD conditions too strict: they enforce eager evaluation of arguments

2. Axiomatic Operators

- Finding model for operator from axioms usually impossible
- Require providing ProB kernel with information on how to represent the operator
- Currently: support added for transitive closure; mapping it to “classical B” version
- Annotation file in Rodin Project: maps Theory operators to ProB Prolog predicates

Constraint-Solving + Theories

```
context Demo
constants cube isqrt sol1 sol2
axioms
 @axm0 cube=(λx·x∈ℤ|x*x*x)
 theorem @thm0 {y|cube(cube(y))=512}={2} // observe: no domain restriction on y

 @axm1 isqrt = {x↦r | x∈ℕ ∧ r∈ℕ ∧ r*r ≤ x ∧ (r+1)*(r+1)>x} // automatically detected as symbolic in new
ProB; observe: no explicit finitness restriction on x
 theorem @thm1 isqrt(100) = 10
 theorem @thm2 isqrt(10) = 3
 theorem @thm3 isqrt[1..20] = 1..4
 theorem @thm4 ({1↦4, 2↦9, 3↦ 26} ; isqrt) = {1↦2, 2↦3, 3↦5}  // using relational composition as map
 theorem @thm5 {x| isqrt(x)=10} = 100..120  // constraint solving with isqrt; observe: no restriction on x
!

 theorem @thm6 cls({1↦2, 2↦3}) = {1↦2,2↦3, 1↦3}
 theorem @thm7 cls(isqrt)[{16}] = {4,2,1}  // this is the transitive closure of an infinite function !
 theorem @thm8 cls(isqrt)[{1024}] = {1,2,5,32}  // this is the transitive closure of an infinite function
!
 @axm8 sol1 = cls(isqrt)[{100000}]
 theorem @thm_sol1 sol1 = {1,2,4,17,316}
 @axm9 sol2 = {x|x∈1..100 ∧ cls(isqrt)[{x}] = {1,2,4}}
 theorem @thm_sol2 sol2 = 16..24

 theorem @nested {x| x∈1..100 ∧ {y|y∈x..x+10} = {z|z≥x ∧ z<x+11 ∧ ∃v·(v>x)}} = 1..100
end
```

Constraint-Solving + Theories

```
context Demo
constants cube isqrt sol1 sol2
axioms
 @axm0 cube=(λx·x∈ℤ|x*x*x)
 theorem @thm0 {y|cube(cube(y))=512}={2} // observe: r

 @axm1 isqrt = {x↦r | x∈ℕ ∧ r∈ℕ ∧ r*r ≤ x ∧ (r+1)*(r+
ProB; observe: no explicit finitness restriction on x
 theorem @thm1 isqrt(100) = 10
 theorem @thm2 isqrt(10) = 3
 theorem @thm3 isqrt[1..20] = 1..4
 theorem @thm4 ({1↦4, 2↦9, 3↦ 26} ; isqrt) = {1↦2, 2↦
 theorem @thm5 {x| isqrt(x)=10} = 100..120  // constrai
!

 theorem @thm6 cls({1↦2, 2↦3}) = {1↦2,2↦3, 1↦3}
 theorem @thm7 cls(isqrt)[{16}] = {4,2,1}  // this is
 theorem @thm8 cls(isqrt)[{1024}] = {1,2,5,32}  // thi
!
 @axm8 sol1 = cls(isqrt)[{100000}]
 theorem @thm_sol1 sol1 = {1,2,4,17,316}
 @axm9 sol2 = {x|x∈1..100 ∧ cls(isqrt)[{x}] = {1,2,4}}
 theorem @thm_sol2 sol2 = 16..24

 theorem @nested {x| x∈1..100 ∧ {y|y∈x..x+10} = {z|z≥x ∧
end
```

State | LTL Counter-Example

Name	Value
Demo	
cube	λx·(x ∈ ℤ|x * (x * x))
isqrt	{x,r|x ∈ N ∧ (r ∈ N ∧ (r *…
sol1	{1,2,4,17,316}
sol2	{16,17,18,19,20,21,22,23,24}
Formulas	
★ invariants	T
axioms	T
cube = λx·(x ∈ ℤ|x * (x * x))	T
isqrt = {x,r|x ∈ N ∧ (r ∈ N ∧ (r * r ≤…	T
sol1 = closure1(isqrt)[{100000}]	T
sol2 = {x|x ∈ 1 .. 100 ∧ closure1(isqrt)…	T
theorems (on constants)	
{y|y ∈ ℤ ∧ cube(cube(y)) = 512} = {2}	T
isqrt(100) = 10	T
isqrt(10) = 3	T
isqrt[1 .. 20] = 1 .. 4	T
({(1↦4),(2↦9),(3↦26)} ; isqrt) = {(1↦2)…	T
{x|x ∈ ℤ ∧ isqrt(x) = 10} = 100 .. 120	T
(LET (r)=={(1↦2),(2↦3)} IN closure1(r))…	T
closure1(isqrt)[{16}] = {1,2,4}	T
closure1(isqrt)[{1024}] = {1,2,5,32}	T
sol1 = {1,2,4,17,316}	T
sol2 = 16 .. 24	T

Highlights

Theory Plugin support

automatic detection of (potentially) infinite sets

solving over infinite domains

transitive closure for infinite relation

nested quantifiers and set comprehensions

```
context Demo
constants cube isqrt sol1 sol2
axioms
 @axm0 cube=(λx·x∈ℤ|x*x*x)
 theorem @thm0 {y|cube(cube(y))=512}={2} // observe: no domain restriction on y

 @axm1 isqrt = {x↦r | x∈ℕ ∧ r∈ℕ ∧ r*r ≤ x ∧ (r+1)*(r+1)>x} // automatically detected as symbolic in new
ProB; observe: no explicit finitness restriction on x
 theorem @thm1 isqrt(100) = 10
 theorem @thm2 isqrt(10) = 3
 theorem @thm3 isqrt[1..20] = 1..4
 theorem @thm4 ({1↦4, 2↦9, 3↦ 26} ; isqrt) = {1↦2, 2↦3, 3↦5}                    map
 theorem @thm5 {x| isqrt(x)=10} = 100..120  // constraint solvi                  on on x
!

 theorem @thm6 cls({1↦2, 2↦3}) = {1↦2,2↦3,
 theorem @thm7 cls(isqrt)[{16}] = {4,2,1}                                       an infinite function !
 theorem @thm8 cls(isqrt)[{1024}] = {1,2,5                                      e of an infinite functio
!
 @axm8 sol1 = cls(isqrt)[{[illegible]00000}]
 theorem @thm_sol1 sol1 = {1,2,4,17,316}
 @axm9 sol2 = {x|x∈1..100 ∧ cls(isqrt)[{x}] = {1,2,4}}
 theorem @thm_sol2 sol2 = 16..24

 theorem @nested {x| x∈1..100 ∧ {y|y∈x..x+10} = {z|z≥x ∧ z<x+11 ∧ ∃v·(v>x)}} = 1..100
end
```

3. Recursive Functions

- Example: uses ifte - direct definition, choose: axiomatic definition, recursive function definition

```
•ifte  :  ifte(Test : BOOL, Then : A, Else : A)  EXPRESSION  PREFIX
direct definition
 ifte(Test : BOOL, Then : A, Else : A) ≙ COND(Test=TRUE,Then,Else)
```

```
CONTEXT
  RecursiveSetFunctions
CONSTANTS
  sumset
AXIOMS
  axm1   :   sumset ∈ ℙ(ℤ) ↔ ℤ
             sumset = (λs·s∈ℙ(ℤ)|
  axm2   :              ifte(bool(s=∅),
                             0,
                             choose(s)+sumset(s\{choose(s)}) ) )
  thm1   :   sumset(∅)=0
  thm2   :   sumset(1‥3)=6
  thm3   :   sumset(1‥1000)=500500
END
```

```
OPERATORS
 •choose:  choose(S : ℙ(A)) EXPRESSION PREFIX  A
well-definedness condition
  S≠∅
AXIOMS
 axm1:  ∀SS·SS⊆A ⇒choose(SS)∈SS
```

Name	Value
▼RecursiveSetFunctions	
sumset	λs·(s ∈ ℙ(ℤ)|{z_,z__|…
▼Formulas	
★ invariants	T
▼axioms	T
▶sumset = λs·(s ∈ ℙ(ℤ)|{z_,z_…	T
▼theorems (on constants)	
▶sumset(∅) = 0	T
▶sumset(1 ‥ 3) = 6	T
▼sumset(1 ‥ 1000) = 500500	T
▶sumset(1 ‥ 1000)	500500

4. Inductive Datatypes + Recursive Operators

```
THEORY
          //   A theory of inductive lists.
  List    //   @author Issam Maamria
          //   @date 06/08/2011
TYPE PARAMETERS
  T
  S
DATATYPES
  List(T)  ≜
   ▶ nil
   ▶ cons(head:T, tail:List(T))
OPERATORS
  •listSize  :  listSize(l : List(T))  EXPRESSION  PREFIX
  recursive definition
   case l
    listSize(nil ) ≜ 0
    listSize(cons(x, l0)) ≜ 1+listSize(l0)
  •append  :  append(l : List(T), x : T)  EXPRESSION  PREFIX
  recursive definition
   case l
    append(nil, x) ≜ cons(x,nil)
    append(cons(x0,l0), x) ≜ cons(x0,append(l0,x))
  •rev  :  rev(l : List(T))  EXPRESSION  PREFIX
  recursive definition
   case l
    rev(nil) ≜ nil ∶ List(T)
    rev(cons(x,l0)) ≜ append(rev(l0),x)
```

```
event do_rev
   then
     @act1 l ≔ rev(l)
     @act2 elems ≔ {a,b · a↦b∈elems ∣ size−1−a↦b}
 end
```

State | LTL Counter-Example

Name	Value	Previous value
ListTest1		
elems	{(0↦2),(1↦3)}	{(0↦3),(1↦2)}
l	cons((2↦cons((3↦nil))))	cons((3↦cons((2↦nil))))
size	2	2
Formulas		
invariants	T	T

Inductive Datatypes and Recursive Operators

- Z Free types close match to Rodin inductive data types; support added to ProB (aka ProZ)
- Recursive functions supported by ProB
- Limited constraint solving added (Demo); but enumeration of inductive datatype will almost always result in “enumeration warning”

Future Work

- Reals / Rationals/ Floats
- Better constraint solving for inductive data types
- Replacing custom Prolog code for axiomatic operators by propagation rules (like proof rules)
- Performance for recursive operators (translating to compiled Prolog, JIT?)
- Rodin: easy way to add definitions and recursive functions

ProB Tcl/Tk REPL

```
Logged on Tue Jan 14 18:28:50 CET 2014
>>>> isqrt(20000)
  141

>>>> isqrt[1..20]
  {1,2,3,4}



>>>> ([1,2,3,4,5,6,7] ; isqrt)
  {(1|->1),(2|->1),(3|->1),(4|->2),(5|->2),(6|->2),(7|->2)}

>>>> isqrt(x) = 100
  TRUE

  ( x=10000 )

>>>> isqrt(x) = 101
  TRUE

  ( x=10201 )

>>>> {x|isqrt(x) = 101}
  #203:{10201,10202,...,10402,10403}


>>>> icls({1|->2, 2|->3})
  {(1|->2),(1|->3),(2|->3)}

>>>> icls(%x.(x:NATURAL|x/2))
  closure1(%x.(x : NAT|x / 2))

>>>> icls(%x.(x:NATURAL|x/2))[{100}]
  {0,1,3,6,12,25,50}
```

```
>>>> cube(cube(y)) = 262144
  TRUE

  ( y=4 )

>>>> cube(cube(y)) > 262143
  TRUE

  ( y=4 )

>>>> icls(isqrt)[{262144}]
  {1,2,4,22,512}

>>>> icls(isqrt)[{2**16}]
  {1,2,4,16,256}

>>>> icls(isqrt)[{x}] = {1,2}
  TRUE

  ( x=4 )

>>>>
```