Practical Theory Extension

Asieh Salehi[1], Michael Butler[1], Jean-Raymond Abrial[2]

University of Southampton[1]

Marseille[2]

5th Rodin Workshop

2nd June 2014, Toulouse

Tutorial Overview

- Part I, Asieh Salehi
  - Motivation
  - Introducing Theory Extension capabilities
  - Simple demo of the Theory plug-in
- Part II, Michael Butler
  - Inductive definitions and proofs
  - Axiomatic definitions
  - Wrapping compound structures in data types
  - Hierarchical file example
- Part III, Jean-Raymond Abrial
  - Well-Definedness, Fixpoint, Closure, Computation, Well-Ordering Theorem, Cantor-Bernstein Theorem, Axiomatisation of Real-Numbers

Motivation: Mathematical theories in Event-B

- Core Rodin supports rich mathematical theories:
  - integers, sets, relations, functions …
- But many problems require additional mathematical structures (e.g., lists, trees, graphs, reals)
- These structures could be defined axiomatically (as an Event-B *context*), **but**
  - polymorphism is not supported
  - no direct way to extend the provers of Rodin
  - no direct support for ensuring soundness of new operator definitions and proof rules

Theory extension plug-in

- Allow users to define new mathematical operators and data types
- Allow users to add proof rules to Rodin prover
  - Rules may be used interactively
  - Rules may be added to automated tactics
- Generate soundness POs for new definitions and proof rules

Forms of definition

- Data types
  - Inductive polymorphic data types (e.g., lists, trees, …)
  - Axiomatic types (e.g., reals)

- Polymorphic operator definitions
  - Direct definitions (e.g., sequences as integer functions)
  - Recursive definitions (e.g., operators on lists, trees, …)
  - Axiomatic definitions (e.g., axioms of real arithmetic)

Sequence Theory

```
THEORY
  Seq      //  A theory of sequences defined as finite partial functions.
TYPE PARAMETERS
  A
OPERATORS
  •seq  :  seq(a : ℙ(A))  EXPRESSION  PREFIX        //  The sequence operato
  direct definition
   seq(a : ℙ(A)) ≙ {n, f · n ∈ ℕ ∧ f ∈ 1‥n→a | f}      //  a set of finite
  •seqSize  :  seqSize(s : seq(A))  EXPRESSION  PREFIX        //  size of s
  direct definition
   seqSize(s : seq(A)) ≙ card(s)
  •seqIsEmpty  :  seqIsEmpty(s : ℤ↔A)  PREDICATE  PREFIX        //  predi
                                                                 //  wheth
  direct definition
   seqIsEmpty(s : ℤ↔A) ≙ seqSize(s)=0
  •emptySeq  :  emptySeq  EXPRESSION  PREFIX        //  empty sequence
  direct definition
   emptySeq≙ ∅ ⦂ ℙ(A)
  •seqHead  :  seqHead(s : seq(A))  EXPRESSION  PREFIX        //  the head
  well-definedness condition
   ¬ seqIsEmpty(s)
  direct definition
   seqHead(s : seq(A)) ≙ s(1)
```

Forms of proof rule

- Theorems (most general form)
  - Theorems can be instantiated manually in proof giving rise to additional hypotheses
- Conditional rewrites:

  lhs = rhs1, if C1

  lhs = rhs2, if C2

  - Can be used manually or automatically
- Inference rules:
  - **Given** P1, P2, …   **Infer** Q
- Induction: available for inductive types

Demo
Proof rules for Sequences

Job Queue Machine

```
MACHINE
      JobQueue
SEES
       C1
VARIABLES
      queue
      job
INVARIANTS
      inv1 :      job ⊆ JOB
      inv2 :      queue ∈ seq(job)      // elements of queue are from job
EVENTS
      INITIALISATION:
      THEN
      ○     act1:       queue ≔ emptySeq ›
      ○     act2:       job ≔ ∅ ›
      END
```

Soundness POs for proof rules

- Theorems:
  - Soundness PO: theorem provable (from definitions and existing theorems)
- Conditional rewrites:
  - Soundness PO: $C1 \Rightarrow lhs = rhs1, \ldots$
- Inference rules:
  - Soundness PO: $P1 \wedge P2 \wedge \ldots \Rightarrow Q$

Proof of seq monotonic

- Demo

Inductive datatypes

- Peano
- List
- Trees

Visibility and Scoping: Deploying a theory

- An individual theory consist of a collection of definitions and proof rules
- The first step to make the definitions and proof rules available to use is deploying a theory.

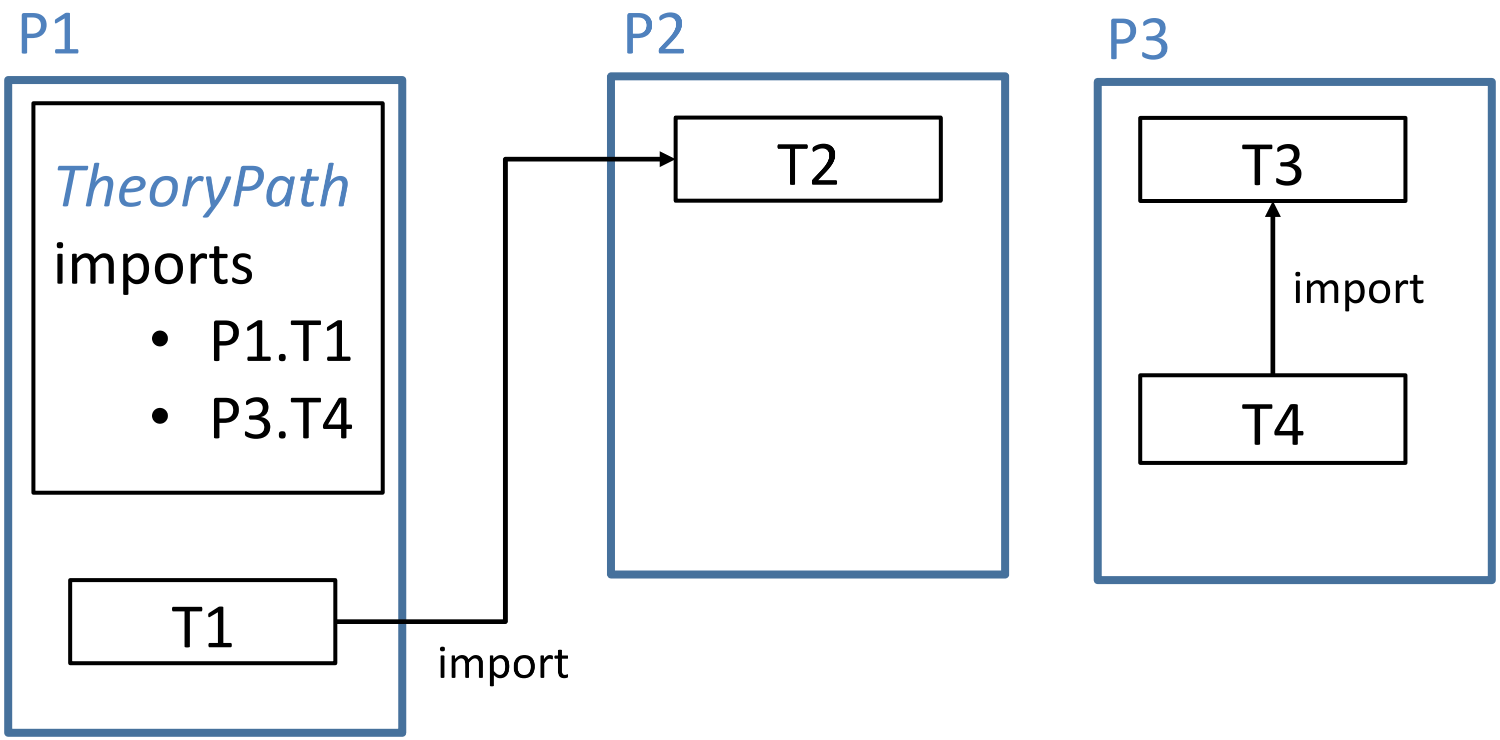
Visibility and Scoping: Importing other theories

- A theory can import other deployed theories
- Hierarchy of theories:
  - Importing theory inherits definitions from the imported theory
  - Importing theory can extend the parent definitions
  - Conflicting validation applied (e.g., same name elements is not allowed)
- A theory can import a local or global theory

Visibility and Scoping: Theory Path

- A *theorypath* is a tool to introduce the deployed theories in a project scope
- A *theorypath* can imports deployed (local/global) theories
- A machine/context accesses (local/global) theories imported directly or indirectly by a *theorypath* within the same project as the machine/context

Visibility and Scoping: Example



Theories T1, T2, T3 and T4 are visible in P1 via *TheoryPath*

Visibility and Scoping: Colour Coding

- White  : a new and un-deployed theory
  - A white theory is not accessible to be imported either in another theory or in a theorypath
- Green  : a deployed updated theory
  - A green theory is deployed and updated.
- Amber  : a deployed out-dated theory
  - an amber theory is modified after deployment; the deployed version of the theory is not sync with the current state of the theory.

Tree structured file store

- Objects can be files or directories
- Unique *root* object
- Each object has a parent (except *root*)
- No loops in the structure
- Each object reachable from *root*
- Operations:
  - create object,
  - add object,
  - delete object (incl directory),
  - move object (incl directory),
  - copy object (incl directory)

No-loop property

1. @inv1: parent $\in$ objects \ {root} $\to$ objects
2. @inv2: parent* $\cap$ id = { }

Invariant 2 is not easy to work with.
Instead we use *Well-foundedness*:

$$wf(R) \;\; == \;\; \forall s \,.\, s \subseteq R^{-1}[s] \;\Rightarrow\; s = \{\,\}$$

3. @inv3: wf(parent)

inv2 becomes a theorem that follows from inv3.

Graph based tree theory

- Tree structure represented by
  - Set of nodes n
  - Root node r
  - Parent function p
- Wrap these as a data type (polymorphic on nodes)
  - TreeType(nodes:n, root:r, parent:p)
- Define a validity predicate
  - ValidTree(t) ==

    parent injective on nodes

    no loops in parent

    root is the ancestor of all other nodes
- Define operators on tree structures
  - addChild, addSubtree, …
- Use theory to specify a machine model of a file system

Concluding

- Summary:
  - Added support for user-defined theories and proof rules in seamless way with soundness POs

- Usage scenarios:
  1. Types and operators identified and defined
  2. Basic proof rules identified and proved
     - soundness POs can uncover errors in definitions and rules
  3. Usage of new theories in models identifies need for additional proof rules
     - These are added to the theories