Putting Theories Together to Make Specification

Bingzhou Zheng

March 31, 2010

Why are we interested in **theories**?

- We need a theory to specify a problem before we can develop a program to solve it.
- We need a theory to represent knowledge.

Why are we interested in **structured descriptions of theories**?

- We find theories very hard to understand unless they have a well-structured description.
- We attempt to represent knowledge as collections of separate simple fragments.

Theory

The paper of Burstall and Goguen (1977) makes theories many-sorted, and with provision for errors.

A many-sorted theory is given by:

- a set of sorts (including the sort of truth value)
- a set of operators (including **true** and **false**)
- a set of laws which the operators above must satisfy

Note:

- The laws take the form of equations with free variables but no quantifiers, which are implicitly universally quantified.
- The equations are closed under inference by reflexivity, transitivity and symmetry of equality, and by substitution.

Language "Clear"

In Burstall and Goguen' paper (1977), they proposed the language "**Clear**" as a tool

- to describe program specification
- to serve to represent knowledge in a machine manipulable form.

The language Clear uses the following theory-building operations and mechanism to build theories.

Theory-Building Operations

To build a structured theory, we must build our theories up from small intelligible pieces:

- the ability to write small and explicit theories, like

```
theory sorts ···
       opns  ···
       eqns  ···  endth
```

- four operations on theories enable us to build up theory expressions denoting complex theories
  - combine
  - enrich
  - induce
  - derive

Four Operations / Combine

```
Nat0
theory sorts nat
       opns  0 : → nat
             succ : nat → nat
       eqns  endth

Bool0
theory sorts bool
       opns  true : → bool
             false : → bool
             ¬ : bool → bool
             ∧ : bool → bool
       vars  p : bool
       eqns  ¬true = false
             ¬false = true
             false ∧ p = false
             true ∧ p = p endth
```

Four Operations / Combine Cont.

+ is used as the combine operation

```
Bool0 + Nat0
theory sorts bool, nat
       opns  true :→ bool
             false :→ bool
             ¬ : bool → bool
             ∧ : bool → bool
             0 :→ nat
             succ : nat → nat
       vars  p : bool
       eqns  ¬true = false
             ¬false = true
             false ∧ p = false
             true ∧ p = p
       endth
```

Four Operations / Enrich

The whole express below expression denotes the new enriched theory.

Nat1

enrich Bool0 + Nat0 by

opns $\leq : nat, nat \to bool$

$eq : nat, nat \to bool$

vars $m, n : nat$

eqns $0 \leq n = true$

$succ(m) \leq 0 = false$

$succ(m) \leq succ(n) = m \leq n$

$eq(m, n) = m \leq n \wedge n \leq m$

endth

Four Operations / Induce

```
Nat
induce enrich Nat0 + Bool by
              opns   ≤ : nat, nat → bool
                     eq : nat, nat → bool
              vars   m, n : nat
              eqns   0 ≤ n = true
                     succ(m) ≤ 0 = false
                     succ(m) ≤ succ(n) = m ≤ n
                     eq(m, n) = m ≤ n ∧ n ≤ m
          endth
```

Induce enable theory to extend the equations of a theory

- an equation holds for a variable n, if it holds for every equation obtained by substituting a variable-free term
- to find equations holding in a theory created by induce, we may prove by induction on the structure of terms

Four Operations / Derive

```
Nat
derive
        sort   element ,  bool
        opns   equal , true , false
from Nat1 by
        eqns   element is nat
               bool is bool
               equal is eq
               true is true
               false is false
        endth
```

Theory Constant / Theory Procedure / Local Theory Definition

Besides the four primitive operations on theories, we should enable the users to define his own operations. So we introduce:

- Theory constant
  - it enables users to give a name to a theory
- Theory procedure
  - it takes theory as their parameters and producing a theory as a result.
  - its body use the primitive operations, and may call other theory procedures (but no recursion).
- Local theory definition
  - it defines a theory in the body of theory procedures

Theory Constant / Local Theory Definition

```
const Nat =
  induce let Nat0 =
          theory sorts nat
                 opns  0 : nat
                       succ : nat → nat
           endth
        in enrich Nat0 + Bool by
          opns  ≤ : nat, nat → bool
                eq : nat, nat → bool
          vars  m, n : nat
          eqns  0 ≤ n = true
                succ(m) ≤ 0 = false
                succ(m) ≤ succ(n) = m ≤ n
                eq(m, n) = m ≤ n ∧ n ≤ m
        endth
```

Theory Procedure

Procedures can only accept a certain sort of theory as parameter.

We use "meta-sort" as the formal parameter of a procedure. The meta-sort is itself a theory, which denotes theories with a certain property.

The actual parameter theory must include all the equations of the meta-sort theory as rewritten under this operator to operator function

Example:

Suppose we have some partially ordered set, and we can form strings from its elements. Now we can develop a theory of ordered strings for any partially ordered set. The partially ordered set can be regarded as a theory parameter. The meta-sort should be a theory of partial orderings.

Theory Procedure Cont.

We can form strings from the elements of any set, so here the meta-sort of string procedure is trival.

```
const Triv = theory sorts element endth
```

proc Strings (X: Triv)=
    induce enrich X by
        sorts *string*
        opns $unit : element \longrightarrow string$
            $\curlywedge \; : \longrightarrow string$
            $\centerdot : string, string \longrightarrow string$
        vars $s, t, u : string$
        eqns $\curlywedge \centerdot s = s$
            $s \centerdot \curlywedge = s$
            $(s \centerdot t) \centerdot u = s \centerdot (t \centerdot u)$
    endth

Theory Procedure Cont.

When we apply Strings procedure to Nat to get strings of natural number, we need to associate the sorts and operators of the meta-sort (Triv) of the formal parameter with those of the actual parameter (Nat).

We need a sort to sort function and an operator to operator function just as in derive.

```
strings ( Nat[element is nat] )
```

Theory Procedure Cont.

To develop a theory of ordered strings, we need write the procedure for ordered strings. This procedure need the theory of partial order for use as a meta-sort.

```
const Poset =
    enrich Bool by
        sorts element
        opns  ≤ : element, element → bool
              eq : element, element → bool
        vars  x, y, z : element
        eqns  x ≤ x = true
              x ≤ y ∧ y ≤ z ∧ ¬(x ≤ z) = false
              eq(x, y) = x ≤ y ∧ y ≤ x
        endth
```

Theory Procedure Cont.

```
proc OrderedStrings (P: Poset)=
   induce enrich Strings(P) by
            opns  ordered : string → bool
            vars  x, y : element
                  s, t, u : string
            eqns  ordered(⅄) = true
                  ordered(unit(x)) = true
                  ordered(unit(x), unit(y)) =
                    x ≤ y
                  ordered(s . t . u) =
                    ordered(s.t)∧ordered(t.u)
         endth
```

```
OrderedStrings
      ( Nat [element is nat,≤ is ≤, eq is eq] )
```

Reference

PUTTING THEORIES TOGETHER TO MAKE SPECIFICATION, R.M.Burstall and J.A.Goguen (1977)

THE SEMANTICS OF CLEAR, A SPECIFICATION LANGUAGE, R.M.Burstall and J.A.Goguen (1979)

Questions?