Till Mossakowski
Lutz Schröder

Software spezification in CASL
Exercise sheet 3

Abgabe Mo., 05.02.06, 10:15 in der Veranstaltung,
oder bis So. 04.02. in Pf. 99

Exercise 1:

Consider the following specifications:

spec STRICT_PARTIAL_ORDER =
  sort *Elem*
  pred __<__ : *Elem* × *Elem*
  ∀ *x*, *y*, *z* : *Elem*
  • ¬ *x* < *x* %(strict)%
  • *x* < *y* ⇒ ¬ *y* < *x* %(asymmetric)%
  • *x* < *y* ∧ *y* < *z* ⇒ *x* < *z* %(transitive)%
  %{ Note that there may exist x, y such that
    neither x < y nor y < x.}%
end

spec STRICT_TOTAL_ORDER =
  STRICT_PARTIAL_ORDER
then ∀ *x*, *y* : *Elem* • *x* < *y* ∨ *y* < *x* ∨ *x* = *y* %(total)%
end

spec PARTIAL_ORDER =
  STRICT_PARTIAL_ORDER
then pred __<=__(*x*, *y* : *Elem*) ⇔ *x* < *y* ∨ *x* = *y*
end

spec PARTIAL_ORDER_1 =
  PARTIAL_ORDER
then %implies
  ∀ *x*, *y*, *z* : *Elem* • *x* <= *y* ∧ *y* <= *z* ⇒ *x* <= *z* %(transitive)%
end

spec TOTAL_ORDER_AUX =
  PARTIAL_ORDER_1
and STRICT_TOTAL_ORDER
end

spec TOTAL_ORDER =

TOTAL_ORDER_AUX **hide** __<__
end

spec NAT =
free type $Nat ::= 0 \mid suc(Nat)$
end

spec NAT_ORDER =
NAT
then pred __<=__ : $Nat \times Nat$
$\forall\ m, n : Nat$
• $0 <= n$ %(leq_def1_Nat)%
• $\neg\ suc(n) <= 0$ %(leq_def2_Nat)%
• $suc(m) <= suc(n) \Leftrightarrow m <= n$ %(leq_def3_Nat)%
end

view V : TOTAL_ORDER **to** NAT_ORDER
end

Prove the view. Use Hets to construct the development graph, but then describe the necessary proof steps using the development graph calculus. You need to construct additional specifications (and corresponding development graph nodes) during the proof.

6 P.

Exercise 2:

Write an architectural specification of a software system that you have encountered in your studies, preferably in the "Softwarepraktikum" or in your student project.

Since the emphasis is on the architectural design, for the structured specifications involved, it suffices if you write down the signatures and some of the axioms. That is, you may omit axioms, but please indicate the omission.

10 P.

Exercise 3:

Why is the following architectural specification ill-formed?

4 P.

```
library Invoice
%% authors: Didier Bert, Hubert Baumeister
%% date: 27.01.03
%% appeared in
%% Software Specification Methods:
%% An Overview Using a Case Study
```

```
%% Marc Frappier, Université de Sherbrooke, Québec, Canada
%% Henri Habrias, Université de Nantes, France,  (Eds.)
%% Published October 2000
%% FACIT, SPRINGER-VERLAG
%% ISBN: 1-85233-353-7

%display __is_in__ %LATEX __\in__

from Basic/Numbers version 0.3 get Nat
from Basic/StructuredDatatypes version 0.3 get List

spec ORDER = Nat
then
  sorts
      Order, Product
  ops
      reference : Order -> Product;
      ordered_qty : Order -> Pos;
  preds
      is_pending, is_invoiced : Order;
  vars  o : Order
    . not is_pending(o) <=> is_invoiced(o)
end

spec STOCK = Nat
then
  sorts
      Stock, Product
  ops
      qty    : Product * Stock ->? Nat;
      add    : Product * Pos * Stock ->? Stock;
      remove : Product * Pos * Stock ->? Stock;
  pred
      __ is_in __ : Product * Stock
  ...
end


spec INVOICE = ORDER and STOCK
then
  free type Msg ::= success | not_pending | not_referenced | not_enough_qty;
  free type OSM ::= mk(order_of:Order; stock_of:Stock; msg_of:Msg);

  pred referenced(o:Order; s:Stock) <=> reference(o) is_in s;
  pred enough_qty(o:Order; s:Stock) <=>
   ordered_qty(o) <= qty(reference(o),s);
```

```
  pred invoice_ok(o:Order; s:Stock) <=>
is_pending(o) /\ referenced(o,s) /\ enough_qty(o,s);

  op invoice_order : Order * Stock -> OSM
  ...
end


spec ORDER_QUEUE =
  { List[ORDER fit Elem |-> Order] with List[Order] |-> OQueue }
then
  pred __is_in__  : Order * OQueue
  ops __ <- __ : OQueue * Order -> OQueue;
      remove   : Order * OQueue -> OQueue;
  ...
end

spec QUEUES = ORDER_QUEUE
then
  preds unicity, pqueue, iqueue : OQueue
  ...
  sorts
      UQueue = {oq:OQueue . unicity(oq)};
      PQueue = {uq:UQueue . pqueue(uq)};
      IQueue = {uq:UQueue . iqueue(uq)};
end

spec WHS = QUEUES and INVOICE
then
  free type GState ::= mk_gs(porders:PQueue; iorders:IQueue; the_stock:Stock);
  op
      the_orders(gs:GState):OQueue = porders(gs) ++ iorders(gs)
  preds
      referenced(oq:OQueue; s:Stock) <=>
forall x:Order . (x is_in oq => referenced(x,s));
      consistent(gs:GState) <=> unicity(the_orders(gs))
/\ referenced(the_orders(gs),the_stock(gs));
  sort
      VGS = {gs:GState . consistent(gs)};

  pred
      invoiceable(pq:PQueue; s:Stock) <=>
exists o:Order . (o is_in pq /\ enough_qty(o,s))
  op  first_invoiceable : PQueue * Stock ->? Order
```

```
  ops
      new_order       : Product * Pos * VGS -> VGS;
      cancel_order    : Order * VGS -> VGS;
      add_qty         : Product * Pos * VGS -> VGS;
      deal_with_order : VGS -> VGS;

      mk_order        : Product * Pos * VGS -> Order

  ...

end


arch spec Warehouse =
units
  NatAlg : Nat;
  OrderFun : Nat -> ORDER;
  OrderAlg = OrderFun[NatAlg];
  StockFun : Nat -> STOCK;
  StockAlg = StockFun[NatAlg];
  InvoiceFun : ORDER * STOCK -> INVOICE given  NatAlg;
  QueuesFun : ORDER -> QUEUES;
  WhsFun : QUEUES * INVOICE -> WHS given OrderAlg, StockAlg;

result
  WhsFun[QueuesFun[OrderAlg]] [InvoiceFun[OrderAlg][StockAlg]]
end
```