Software specification in CASL - Datatypes II

Till Mossakowski, Lutz Schröder

November 2006

SFB/TR 8 SPATIAL COGNITION

Free Datatypes: Natural Numbers

spec $\text{NAT} =$

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

The Signature Corresponding to the Type Declaration

sort Nat

ops $0 : Nat;$

$suc : Nat \longrightarrow Nat$

The Peano Axioms

- no confusion: two different terms denote different elements in the model
- no junk: any element in the model is denoted by some term
- altogether: each model is isomorphic to the model consisting of all terms

No Confusion

Two different terms denote different elements in the model

- each constructor is injective

  $\forall\ x : Nat;\ y : Nat$

  - $suc(x) = suc(y) \Leftrightarrow x = y$ **%**(ga_injective_suc)**%**

- any two constructors have disjoint images

  $\forall\ x : Nat \bullet \neg\ 0 = suc(x)$ **%**(ga_disjoint_0_suc)**%**

No Junk

Any element in the model is denoted by some term.

This corresponds to an induction principle:

$$\forall P : pred(Nat) \bullet \\ \Big(P(0) \wedge \forall x : Nat \bullet P(x) \Rightarrow P(suc(x))\Big) \\ \Rightarrow \forall x : Nat \bullet P(x)$$

The induction principle is a second-order formula and not expressible in first-order logic. CASL admits such second-order formulas; they are called sort generation constraints in CASL. Notation: $(\{Nat\}, \{0; suc\})$.

Lists

spec LIST [**sort** $Elem$] = **%mono**
free type $List[Elem] ::= [] \mid \_\_::\_\_(Elem; List[Elem])$
end

The Signature Corresponding to the Type Declaration

sort $List[Elem]$

op $[] : List[Elem]$

op $\_\_::\_\_ : Elem \times List[Elem] \rightarrow List[Elem]$

No Confusion

- each constructor is injective

  $\forall\ X1 : Elem;\ X2 : List[Elem];\ Y1 : Elem;\ Y2 : List[Elem]$
  $\bullet\ X1 :: X2 = Y1 :: Y2 \Leftrightarrow X1 = Y1 \wedge X2 = Y2$

  %(ga_injective___::__)**%**

- any two constructors have disjoint images

  $\forall\ Y1 : Elem;\ Y2 : List[Elem] \bullet \neg\ [] = Y1 :: Y2$

  %(ga_disjoint_[]___::__)**%**

No Junk

Any element in the model is denoted by some term.

This corresponds the following induction principle:

$$\forall P : pred(List[Elem])\bullet$$
$$\Big(P([]) \wedge \forall x : Elem; l : List[Elem] \bullet P(l) \Rightarrow P(x :: l)\Big)$$
$$\Rightarrow \forall l : List[Elem] \bullet P(l)$$

Sort generation constraint in CASL:
$(\{List[Elem]\}, \{[]; \_\_ :: \_\_\})$.

Strings and Lists of Strings

spec CHAR =
    free type $Char ::= A \mid B \mid \ldots$
end

spec STRING =
  LIST[CHAR **fit** $Elem \mapsto Char$] **with** $List[Char] \mapsto String$
end

spec STRINGLIST =
    LIST[STRING **fit** $Elem \mapsto String$]
end

Mutually Recursive Datatypes

spec UnboundedBranchingTree[**sort** $Elem$] =

free types

$Tree ::= Leaf(Elem) \mid Branch(Forest);$

$Forest ::= Nil \mid Cons(Tree; Forest)$

end

The Signature Corresponding to the Type Declaration

sorts $Forest, Tree$

op $Branch : Forest \to Tree$

op $Cons : Tree \times Forest \to Forest$

op $Leaf : Elem \to Tree$

op $Nil : Forest$

Each constructor is injective. . .

$\forall\ X1 : Elem;\ Y1 : Elem$

$$\bullet\ Leaf(X1) = Leaf(Y1) \Leftrightarrow X1 = Y1$$

%(ga_injective_Leaf)%

$\forall\ X1 : Forest;\ Y1 : Forest$

$$\bullet\ Branch(X1) = Branch(Y1) \Leftrightarrow X1 = Y1$$

%(ga_injective_Branch)%

$\forall\ X1 : Tree;\ X2 : Forest;\ Y1 : Tree;\ Y2 : Forest\ \bullet$

$$Cons(X1,\ X2) = Cons(Y1,\ Y2) \Leftrightarrow X1 = Y1 \wedge X2 = Y2$$

%(ga_injective_Cons)%

. . . any two constructors have disjoint images

$\forall\ X1 : Elem;\ Y1 : Forest$

$\bullet\ \neg\ Leaf(X1) = Branch(Y1)$

%(ga_disjoint_Leaf_Branch)**%**

$\forall\ Y1 : Tree;\ Y2 : Forest$

$\bullet\ \neg\ Nil = Cons(Y1,\ Y2)$ **%**(ga_disjoint_Nil_Cons)**%**

No Junk

The induction principle:

$$\forall P : pred(Tree); Q : pred(Forest) \bullet$$
$$\Big(\forall x : Elem \bullet P(Leaf(x)) \wedge$$
$$\forall f : Forest \bullet Q(f) \Rightarrow P(Branch(f)) \wedge$$
$$Q(Nil) \wedge$$
$$\forall t : Tree; f : Forest \bullet P(t) \wedge Q(f) \Rightarrow Q(Cons(t, f))\Big)$$
$$\Rightarrow \forall t : Tree \bullet P(l) \wedge \forall f : Forest \bullet Q(f)$$

Sort generation constraint in CASL:

$(\{Tree; Forest\}, \{Leaf; Branch; Nil; Cons\})$.