Big operations

Yves Bertot
14 March

MAP INTERNATIONAL SPRING SCHOOL ON FORMALIZATION OF MATHEMATICS 2012

SOPHIA ANTIPOLIS, FRANCE / 12-16 MARCH

Inria informatics mathematics · ihp Institut Henri Poincaré · MICROSOFT RESEARCH INRIA JOINT CENTRE

Iterating binary operations

- Binary operations abound in mathematics
- Big operations generalize to n-ary applications
- Many features or big operations are common
- A systematic treatement of the infrastructure

Notations

- Special cases `\sum_(i < n) f i`, `\prod_(j <= k < n) g i`
- well typed if `f g : nat -> nat`, `f : 'I_n -> nat`
- Possibility to filter `\sum_(i < n | odd i) f i`
- Ways to choose the operator and starting value
  `\big[op/id]_(i < n | P i) f i`
- `\sum_(i < | odd i) f i =`
  `\big[addn/0]_(i < n | odd i) f i`

Ranges

- `\sum_(i < n) F i`
  - the type of `i` in `F i` is `'I_n` : this brings information
  - The elements are taken in increasing order
- `\sum_(i < n | P i) F i`
- `\sum_(i \in odd5) F i` if `odd5` is a collective predicate on a finite type
- `\prod_i F i` if the domain of `F` is a finite type
- `\sum_(m <= i < n) F i`
- `\big[op/v]_(i <- s) F i`
- Finite types, intervals, and sequences come with a natural order

Plain operators

- Some theorems don't rely on any property from the operator
  - Empty ranges : `big_nil`, `big_ord0`, `big_geq`
  - Predicate not satisfied: `big_hasC`, `big_pred0`
  - Detaching the leftmost value : `big_cons`, `big_ltn`
  - Range format switching : `big_nth`
  - Widening range: `big_*widen`, `big_*narrow`
  - Exchanging function and predicate : `eq_big`, `eq_bigl`, `eq_bigr`
  - Look for section `Extensionality` in `bigop.v`

Example

```
Section test.
Variables (op1 : nat -> nat -> nat) (v : nat).

Lemma cmp_op3 : \big[op1/v]_(1 <= i < 3) i = op1 1 (op1 2 v).
rewrite big_ltn.
 ================
 op1 1 \big[op1/v]_(2 <= i < 3) i = op1 (op 2 v).

subgoal 2 is: 1 < 3
rewrite big_ltn; last by []
 ================
 op1 1 (op1 2 \big[op1/v]_(3 <= i < 3) i) = op1 (op2 v)

rewrite big_geq; last by []
op1 1 (op1 2 v) = op1 1 (op1 2 v)
```

Monoid structures

- Cut range in two, start from the right,
  - big_cat, big_cat_nat
  - big_nat_recr, big_ord_recr, only without filter
- Replacing all absent elements with the neutral
  - big_mkcond

```
big_mkcond : forall ... ,
  \big[*%M/1]_(i <- r | P i) F i =
     \big[*%M/1]_(i <- r) (if P i then F i else 1).
```

Example with monoid structures

```
Lemma s3' : \sum_(i < 3) i = 3.
rewrite big_ord_recr.
================
addn_monoid
     (\big[addn_monoid/0]_(i < 2)
         widen_ord (m:=3) (leqnSn 2) i)
     ord_max = 3
rewrite big_ord_recr /=.
================
   \sum_(i < 1) i + 1 + 2 = 3
```

Abelian structures

- Divide arbitrarily, partition, re-order, pick one element

```
big_split : forall ... (op : Monoid.com_law idx) ...,
      \big[op/idx]_(i <- r | P i) op (F1 i) (F2 i) =
      op (\big[op/idx]_(i <- r | P i) F1 i)
        (\big[op/idx]_(i <- r | P i) F2 i)
```

- Exchange big operations

```
exchange_big : forall ...,
  \big[op/idx]_(i | P i) \big[op/idx]_(j | Q j) F i j =
    \big[op/idx]_(j | Q j) \big[op/idx]_(i | P i) F i j.
```

Example with re-indexing

```
Lemma sumnP : forall n, \sum_(i < n) i = (n * n.-1) %/2.
suff <- : 2 * \sum_(i < n) i = n * n.-1 by rewrite mulKn.
Continue in a demonstration!!
```

Distributivity

- Distributivity concerns the exchange of two operations
- Multiplication by a scalar, but also by a big sum.

$$(a_{1,1} + a_{1,2} + a_{1,3})(a_{2,1} + a_{2,2} + a_{2,3}) =$$
$$\sum_{f \in \{1,2,3\}^{\{1,2\}}} \prod_{i \in \{1,2\}} a_(i, f(i))$$

- We can range over all functions because it is also a fintype.
- Scalar: `big_distrr`, sums: `big_distr_big`
- Also with dependent choices

Properties

- Properties satisfied by elements and preserved by operators are satisfied

```
big_prop : forall ... ,
  Pb idx ->
  (forall x y : R, Pb x -> Pb y -> Pb (op1 x y)) ->
  (forall i : I, P i -> Pb (F i)) ->
  Pb (\big[op1/idx]_(i <- r | P i) F i)
```

- Similar theorem `big_rel` to relate two big operations
- Advised use: `elim/big_prop: _` and `elim/big_rel: _`
- Caveat: the name of these theorems will change in future versions of SSReflect.

Morphisms

- When phi is a morphism between two monoid structures

```
big_morph: forall ... ,
   {morph phi : x y / op1 x y >-> op2 x y} ->
    phi idx1 = idx2 ->
   phi (\big[op1/idx1]_(i <- r | P i) F i) =
    \big[op2/idx2]_(i <- r | P i) phi (F i)
```

- Demonstration if time allows

think big

- Available for any list, binary operation, and value
- Specific theorems require specific properties
  - `big_nat_recr` requires associativity
- Properties are attached to operators using canonical structures
  - For associativity: Monoid.law.
    Canonical Structure op2Mon : Monoid.law 0 :=
        Monoid.Law op2A op20n op2n0.
  - `op2A`, `op20n` and `op2n0` would have to be proofs that some operation (`op2`) is associative and that some element (0) is left neutral and right neutral for this operation.
- Demonstration if time allows.