Big operations

Yves Bertot
14 March
Iterating binary operations

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

- Special cases \( \sum_{i < n} f_i, \prod_{j \leq k < n} g_i \)
- well typed if \( f, g : \text{nat} \rightarrow \text{nat}, f : \text{'I}_n \rightarrow \text{nat} \)
- Possibility to filter \( \sum_{i < n | \text{odd } i} f_i \)
- Ways to choose the operator and starting value
  \( \big[op/id]\)_{(i < n | P i)} f_i
- \( \sum_{i < n | \text{odd } i} f_i = \big[\text{addn}/0]\)_{(i < n | \text{odd } i)} f_i
Ranges

- $\sum (i < n) F i$
  - the type of $i$ in $F$ is $'I\_n$: this brings information
  - The elements are taken in increasing order
- $\sum (i < n \mid P i) F i$
- $\sum (i \in \text{odd5}) F i$ if $\text{odd5}$ is a collective predicate on a finite type
- $\prod i F i$ if the domain of $F$ is a finite type
- $\sum (m \leq i < n) F i$
- $\big[op/v]_\text{op/v}(i <- s) F i$

Finite types, intervals, and sequences come with a natural order
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
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`

```latex
big_mkcond : forall \ldots , \\
\big[\star M/1\big]_\ldots(i \leftarrow r \mid P i) F i = \\
\big[\star M/1\big]_\ldots(i \leftarrow 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

\[
\text{big\_split : forall ... (op : Monoid.com\_law \ idx) ...,}
\]
\[
\begin{align*}
&\big[op/idx]\_\(i <- r \mid P \ i\) \ op \ (F1 \ i) \ (F2 \ i) = \\
&\ op \ (\big[op/idx]\_\(i <- r \mid P \ i\) \ F1 \ i) \\
&\ (\big[op/idx]\_\(i <- r \mid P \ i\) \ F2 \ i) \\
\end{align*}
\]

- Exchange big operations

\[
\text{exchange\_big : forall ... ,}
\]
\[
\begin{align*}
&\big[op/idx]\_\(i \mid P \ i\) \ \big[op/idx]\_\(j \mid Q \ j\) \ F \ i \ j = \\
&\ \big[op/idx]\_\(j \mid Q \ j\) \ \big[op/idx]\_\(i \mid P \ i\) \ F \ i \ j.
\end{align*}
\]
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.

\[(a_{1,1} + a_{1,2} + a_{1,3})(a_{2,1} + a_{2,2} + a_{2,3}) = \sum_{f \in \{1,2,3\} \setminus \{1,2\}} \prod_{i \in \{1,2\}} a(i, f(i))\]

- 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

\[
\text{big\_prop} : \forall \ldots, \\
\text{Pb idx} \rightarrow \\
(\forall x, y : R, \text{Pb x} \rightarrow \text{Pb y} \rightarrow \text{Pb (op1 x y)}) \rightarrow \\
(\forall i : I, \text{P i} \rightarrow \text{Pb (F i)}) \rightarrow \\
\text{Pb (\big[\text{op1/idx}\]_\text{(i <- r | P i) F i})}
\]

- Similar theorem \text{big\_rel} to relate two big operations
- Advised use: \text{elim/big\_prop: _ and elim/big\_rel: _}
- Caveat: the name of these theorems will change in future versions of \text{SSReflect}.
Morphisms

- When $\phi$ is a morphism between two monoid structures

$$\text{big\_morph: } \forall \ldots, \{\text{morph } \phi : x y \rightarrow op1 x y \rightarrow op2 x y\} \rightarrow \phi \ idx1 = idx2 \rightarrow \phi (\big[op1/idx1] (i \leftarrow r \mid P\ i) F\ i) = \big[op2/idx2] (i \leftarrow r \mid P\ i) \phi \ (F\ i)$$

- 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`.
    ```lean
    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.