Partial co-recursive functions

Yves Bertot
INRIA Sophia Antipolis

Inductive to Co-inductive types

- Inductive types
  - Constructors, pattern-matching,
  - Recursive programming parallels proofs by induction,
  - No general recursion, but close (accessibility),
  - Support partial recursion (thanks to dependent types).
- Infinite data-types in functional programming!
- Coinductive types
  - pattern-matching, constructors,
  - A dual form of recursion (co-recursion),
  - Partiality?

Recursion

- Introduction: 0, $Succ$ (constructors),
- Elimination:
  $rec_{N,A} : A \to (A \to A \to A) \to A \to A$,
  - $rec_{N,A}\ 0\ v\ f = 0$
  - $rec_{N,A}\ (Succ\ p)\ v\ f = f\ p\ (rec_{N,A}\ p\ v\ f)$,
- Dépendance:

$$rec_{N,P} : \begin{array}{l} P(0) \to \\ (\forall x, P(x) \to P(Succ(x))) \to \\ \forall x, P(x) \end{array}$$

- Recursion is controlled in the way it *consumes* inductive data,
- References (Girard, Martin-Löf),

Guarded recursion

- A writing style improvement,
- Pattern matching:

$$\begin{aligned} g_1\ 0 &= a \\ g_1\ (Succ\ x) &= h\ x \end{aligned}$$

- Recursion, g_2 is applied on a subterm:

$$\begin{aligned} g_2\ 0 &= a \\ g_2\ (Succ\ x) &= f\ x\ (g_2\ x) \end{aligned}$$

- Partial functions, $(inv \ldots)$ must be a subterm:

$$\begin{aligned} g_3\ 1\ dom_1 &= a \\ g_3\ x\ dom_x &= f\ x\ (g_3\ \ (x/2) \\ &\qquad\qquad (inv\ x\ dom_x\ xneq1)) \end{aligned}$$

Co-recursion

- observers come first, Constructors come later,
- lists: $hd : L \to A$ and $tl : L \to L$ as observers,
- $corec_{L,B} : B \to (A \times B) \to B \to L$,
  - $hd(corec_{L,B}\ f\ b) : \pi_1(f\ b)$,
  - $tl(corec_{L,B}\ f\ b) : \pi_2(f\ b)$.
- Duality: rec applied to a constructor is a redex, an observer applied to $corec$ is a redex.

Guarded corecursion

- Observers as pattern matching constructs on a set of constructors,
- Infinite lists: one constructor $cons : A \to L \to L$,
  - $hd(cons\ a\ b) = a$, $tl(cons\ a\ b) = b$,
- Use in recursive construction:
  $gb = cons\ (f_1\ b)\ (g\ (f_2\ b))$
- Guard: g can only appear as second argument of $cons$,
- A syntactic criterion to ensure *productivity*.

Partial co-recursion

- Problem: several recursive calls may be needed before producing data,
- Favourite example: $filter$ on infinite lists (streams),
- Only some input streams ensure that filter produces a stream.
- Beyond the expressive power of guarded co-recursion?

Solution using dependent types

- Add a second argument expressing that the first one is in the domain,
- The domain definition has a co-inductive part and an inductive part,
- Non-productive recursive calls are guarded w.r.t. inductive par of the domain.

Example on filter

- A stream is a good argument for $filter$ if:
  - it contains an acceptable element (inductive predicate),
  - its tail also is a good argument (co-inductive predicate).
- Structure of $filter$:
  - A recursive part to compute three items
    - first acceptable element,
    - stream tail after that element,
    - proof that the tail is in the domain,
  - A co-recursive part restarts the search on the tail after producing the element.

Applications

- Eratosthenes' sieve on streams of natural numbers,
- Exact arithmetic on continued fractions (again streams of natural

```
sieve (p:rest) =
    p:(sieve
        [y | y<-rest,
             y `rem` p /= 0])
primes = sieve [2..]
```

- Possibility to separate domain computation from "informative computation" numbers).