Schemata of Recursive Functions and Iterative Algorithms

Dominique Cansell (Lessy, EBRP)

dominique.cansell@gmail.com

June 2024

Outline

- JRA proposed a new instantiation context in 2020

- A tool was developed in the EBRP project (L. Voisin, G. Verdier)

- No machine instantiation (for the moment)

- Explanation on recursive functions

- I used recursive schemata to teach recursion (1990-2000)

Principle very simple

- A machine can see a context
- If we instantiate a context we can have the instantiated machine for free
- Let be a context which defines a total order on a set V
- Let be a refinement chain of a sort algorithm. The last one can be transform to an algorithm.
- If you instantiate the context with a concrete relation. If you prove (Instantiated PO) that the relation is a total order. You can have an sort algorithm for free.

Theorems between Closure an Well-founded Relation

Let r be a relation ($r \in S \leftrightarrow S$) then its transitive closure is defined by a fixpoint and

- if r is well-founded then $closure(r)$ is also well-founded and $\forall x \cdot x \in S \Rightarrow x \notin closure(r^{-1})[\{x\}]$

- if r is well-founded and $r^{-1} \in S \nrightarrow S$ then $finite(closure(r^{-1})[\{x\}]))$

Recursive functions are defined with a well-founded relation and the fixpoint theorem.

- Let r be a well-founded relation on S: $r \in S \leftrightarrow S$
- Let g be a function such that: $g \in (S \times (S \nrightarrow T)) \rightarrow T$
- There is a unique total function fr: $fr \in S \rightarrow T$

  such that we have:

$$\forall x \cdot x \in S \;\Rightarrow\; fr(x) = g(x \mapsto r^{-1}[\{x\}] \lhd fr)$$

More on g: $\forall x, h \cdot x \mapsto h \in dom(g) \Rightarrow r^{-1}[\{x\}] \subseteq dom(h)$

- The value of fr at x depends on its values on the set $r^{-1}[\{x\}]$, $FrSB$ is a function (an operator) which gives the recursive fonction fr:

$$fr \ = \ FrSB(r \mapsto g)$$

Thanks to WD PO when we use $FrSB(R \ \mapsto \ G)$ R must be a well-founded relation and G a function

FrSB means recursive Function (from) S (to) B

Many recursive functions have only one recursive call then $r^{-1}[\{x\}]$ is empty (base case) or a singleton then r^{-1} is a function. In this case we define the function (operator) $FrSB1$ where

$FrSB1(r \mapsto f0 \mapsto f) = FrSB(r \mapsto g)$ and

$$\begin{aligned} g = \{x, h, b \cdot x \in S \wedge h \in S \nrightarrow B \wedge r^{-1}[\{x\}] \subseteq dom(h) \wedge \\ (x \notin ran(r) \Rightarrow b = f0(x)) \wedge \\ (x \in ran(r) \Rightarrow b = f(x \mapsto h(r^{-1}(x)))) \\ \mid x \mapsto h \mapsto b\} \end{aligned}$$

in this case we have the following THEOREMS

$\forall x \cdot x \in S \land x \notin ran(r) \;\Rightarrow\; fr(x) = f0(x)$ and

$\forall x \cdot x \in S \land x \in ran(r) \;\Rightarrow\; fr(x) = f(x \mapsto fr(r^{-1}(x)))$

More classical

A function fr is terminal recursive if

$$fr = FrSB1(r \mapsto f0 \mapsto f) \text{ and } f(x \mapsto y) = y$$

then we have

$$\forall x \cdot x \in S \land x \in ran(r) \Rightarrow fr(x) = fr(r^{-1}(x))$$

The function (operator) $FrSB1Ter$ gives the function :

$$fr = FrSB1Ter(r \mapsto f0) .$$

Let fr equals $FrSB1Ter(r \mapsto f0)$ and $x \in S$, a variable R , an event which computes $fr(x)$ in one shot

$$\text{final} \;=\; \textbf{then}\; R := fr(x)\; \textbf{end}$$

A refinement

Let y be a variable initialised to x with the invariant $fr(x) = fr(y)$

final $=$ **when** $y \notin ran(r)$ **then** $R := f0(y)$ **end**

progress $=$ **when** $y \in ran(r)$ **then** $y := r^{-1}(y)$ **end**

The invariant is: $y \in S \wedge fr(x) = fr(y)$

The variant is trivially $closure(r^{-1})[\{y\}]$ then progress cannot take the control forever.

An algorithm

Using JRA's merging rules we obtain the following algorithm:

$y := x;$
while $y \in ran(r)$ **do**
  $y := r^{-1}(y)$
od;
$R := f0(y)$

An example: gcd with mod

We can define $gcdmod$ with the following definition:

$$gcdmod$$

$$=$$

$$FrSB1Ter(\{a, b \cdot b > 0 \wedge a > b | \\ (b \mapsto a \bmod b) \mapsto (a \mapsto b)\} \\ \mapsto (\lambda x \mapsto y \cdot x > y \wedge y \geq 0 \mid x))$$

After proving that the relation is well-founded we got for free:

$\forall a \cdot a > 0 \;\Rightarrow gcdmod(a \mapsto 0) = a$ and

$$\forall a, b \cdot a > b \wedge b > 0$$

$$\Rightarrow gcdmod(a \mapsto b) = gcdmod(b \mapsto (a \bmod b))$$

A refinement for free

Let ya, yb be variable initialised to xa and xb

final =
  when
    $ya \mapsto yb \notin ran(\{a, b \cdot b > 0 \wedge a > b | (b \mapsto a \bmod b) \mapsto (a \mapsto b)\})$
  then
    $R := (\lambda x \mapsto y \cdot x > y \wedge y \geq 0 \mid x)(ya \mapsto yb))$
  end

progress =
  when
    $ya \mapsto yb \in ran(\{a, b \cdot b > 0 \wedge a > b | (b \mapsto a \bmod b) \mapsto (a \mapsto b)\})$
  then
    $ya \mapsto yb := \{a, b \cdot b > 0 \wedge a > b | (b \mapsto a \bmod b) \mapsto (a \mapsto b)\}^{-1}(ya \mapsto yb)$
  end

with the invariant
$ya \mapsto yb \in \{a \mapsto b | a > b \wedge b \leq 0\} \ \wedge \ gcdmod(xa \mapsto xb) = gcdmod(ya \mapsto yb)$

The variant: $closure(\{a, b \cdot b > 0 \wedge a > b | (b \mapsto a \bmod b) \mapsto (a \mapsto b)\}^{-1})[\{ya \mapsto yb\}]$

A refinement for free

Let ya, yb be variable initialised to xa and xb

final $=$
when
$ya \mapsto yb \notin ran(\{a, b \cdot b > 0 \land a > b | (b \mapsto a \bmod b) \mapsto (a \mapsto b)\})$
then
$R := (\lambda x \mapsto y \cdot x > y \land y \geq 0 \mid x)(ya \mapsto yb))$
end

progress $=$
when
$ya \mapsto yb \in ran(\{a, b \cdot b > 0 \land a > b | (b \mapsto a \bmod b) \mapsto (a \mapsto b)\})$
then
$ya \mapsto yb := \{a, b \cdot b > 0 \land a > b | (b \mapsto a \bmod b) \mapsto (a \mapsto b)\}^{-1}(ya \mapsto yb)$
end

A refinement for free

Let ya, yb be variable initialised to xa and xb

final $=$
when
$yb = 0$
then
$R := ya$
end

progress $=$
when
$yb \neq 0$
then
$ya \mapsto yb := yb \mapsto ya \bmod yb$
end

Can be manage by a refinement step, theorems in the instantiated context or a plugin

An algorithm

Using JRA's merging rules we obtain the following algorithm:

$$
\begin{array}{l}
ya \mapsto yb := \; xa \mapsto xb; \\
\textbf{while } yb \neq 0 \textbf{ do} \\
\quad ya \mapsto yb := yb \mapsto ya \bmod yb \\
\textbf{od}; \\
R := ya
\end{array}
$$

Conclusion

- Instantiate a development is possible

- More schemata on recursive functions are developed

  – with a stack (2 loops)

  – without stack if r is also a function
  (we can calculate the previous value

  – only one loop when $ran(r) = S \setminus \{x_0\}$
  and r is also a function