Inductive Set Construction using Instantiation and Rodin

Dominique Cansell (Lessy, EBRP)

1 Description

In [2] we presented the new Jean-Raymond Abrial's instantiation context to define closure, fixpoint (Tarski), well-foundedness (Noether) and general recursion. A new instantiation plugin [4] was developed in the EBRP project [5]. By early 2021 Jean-Raymond Abrial's new instantiation was functional; however, inductive structures like lists or trees had not been addressed. Jean-Raymond asked me to define these inductive structures using only set theory and his new instantiation method. In this paper we describe an inductive set construction of lists, inductive subset of lists (ordered one) and recursive functions on lists using Rodin and the new instantiation. Rodin [6] is used in March 2021 to develop and prove all the models described in this paper.

2 Well-founded Relation, least Fixpoint and General Recursion

Reference [3] provides complete definitions and proofs for this section.

wf is the set of well-founded relation on S:

$\text{wf} \ = \ \{r \mid r \in S \leftrightarrow S \ \wedge \ (\forall p \cdot p \subseteq r[p] \ \Rightarrow \ p = \varnothing)\}$

fix is the function which gives the least fixpoint on an monotonic function (Knaster-Tarski theorem):

$$\text{fix} \ \in (\mathbb{P}(S) \to \mathbb{P}(S)) \to \mathbb{P}(S)$$

$$\forall\, h \cdot \ h \ \in \mathbb{P}(S) \to \mathbb{P}(S) \ \wedge \ (\forall a, b \cdot a \subseteq b \ \Rightarrow \ h(a) \subseteq h(b))$$
$$\Rightarrow$$
$$h(\text{fix}(h)) \subseteq \text{fix}(h) \ \wedge \ (\forall s \cdot s \subseteq S \ \wedge h(s) \subseteq s \ \Rightarrow \ \text{fix}(h) \subseteq s) \ \wedge \text{fix}(h) = h(\text{fix}(h))$$

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 a function such that: $g \in (S \times (S \nrightarrow T)) \to T$
- There is a unique total function fr: $fr \in S \to T$

such that we have: $\forall x \cdot x \in S \ \Rightarrow \ fr(x) = g(x \mapsto r^{-1}[\{x\}] \lhd fr)$

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

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 $g = \{x, h, b \cdot x \in S \wedge h \in S \nrightarrow B \wedge r^{-1}[\{x\}] \subseteq dom(h) \wedge (x \notin ran(r) \Rightarrow b = f0(x)) \wedge (x \in ran(r) \Rightarrow b = f(x \mapsto h(r^{-1}(x)))) \mid x \mapsto h \mapsto b\}$ in this case we have $\forall x \cdot x \in S \wedge x \notin ran(r) \ \Rightarrow \ fr(x) = f0(x)$ and $\forall x \cdot x \in S \wedge x \in ran(r) \ \Rightarrow \ fr(x) = f(x \mapsto fr(r^{-1}(x)))$

3 How can we define list using set theory and Rodin

Forty years ago we manually defined L the set of lists over A for our students as the smallest set solution of $L = \{()\} \cup \{(a, l) | a \in A \land l \in L\}$. We also define the $empty$ and $cons$ constructors. At that time, students could defines functions on lists abstractly (independently of any Pascal implementation) more easily by using recursion and both constructors. Using set theory $L \mapsto empty \mapsto cons$ is in the set

$\{l \mapsto e \mapsto c | e \in l \land c \in A \times l \rightarrowtail l \setminus \{e\} \land l = \text{fix}(\lambda li | \{e\} \cup c[A \times li])\}$

The problem is as follow: Rodin cannot automatically type this set; we must define a carrier set $ToType$ to initiate the typing process. $List$ is subsequently defined as

$List = \{l \mapsto e \mapsto c \mid l \subseteq ToType \land e \in l \land c \in A \times l \rightarrowtail l \setminus \{e\} \land$
$l = \text{fix}(\lambda li \cdot li \in \mathbb{P}(ToType) | \{e\} \cup c[A \times li])\}$

where the the fix point is instantiated by $S := ToType$. Finally, we can prove the following reformulation of $List$.

$List = \{l \mapsto e \mapsto c \mid l \subseteq ToType \land e \in l \land c \in A \times l \rightarrowtail l \setminus \{e\} \land$
$(\forall X \cdot e \in X \land c[A \times X] \subseteq X \Rightarrow L \subseteq X)\}$

We can easily prove for all $L \mapsto nil \mapsto cons \in List$ the following theorems:

$$(\forall X \cdot nil \in X \land cons[A \times X] \subseteq X \Rightarrow L \subseteq X) \land L = \text{ran}(cons) \cup \{nil\}$$

The first one is the classical induction rule for lists.

It is crucial that $cons$ is an injection. This ensures that if we have two constructed lists that are equals their components must be equal (if $cons(a1 \mapsto l1) = cons(a2 \mapsto l2)$ then $a1 = a2$ (same value) and $l1 = l2$). Since $cons$ is an function from $A \times L$ to $L \setminus \{nil\}$ then if A is empty then $L = \{nil\}$. Conversely, if A is not empty then L cannot be finite. Note that the set $List$ can be empty if $ToType$ is finite; therefore, $ToType$ should not be instantiated with a finite set. At present, no verification for this constraint has been implemented in the plugin. A mathematician will never make such error.

Furthermore when $List$ contains two different elements; there exists an isomorphism between any two sets of lists that preserves the structure provided by constructors.

Important: In the following sections we assume the existence of constants L, nil, $cons$ and $List$ such as

$$L \mapsto nil \mapsto cons \in List\ .$$

4 Simple recursive function on lists

$\{l, a \cdot l \in L \land a \in A | l \mapsto cons(a \mapsto l)\}$ is a well-founded relation on L

$\{l, a, s \cdot l \in L \land a \in A \land s \in S | ((l \mapsto fs(s)) \mapsto (cons(a \mapsto l) \mapsto s)\}$ is a well-founded relation on $L \times S$ where $fs \in S \to S$.

Using $FrSB1$ we can easily define recursive functions on L with $FrLB$ or $L \times S$ with $FrLSB$

$FrLB(f0 \mapsto f) = FrSB1(\{l, a \cdot l \in L \land a \in A | l \mapsto cons(a \mapsto l)\} \mapsto f0 \mapsto f)$ then

$\forall f0, f, fr \cdot f0 \mapsto f \mapsto fr \in FrLB$
$\Rightarrow fr \in L \to B \land fr(nil) = f0(nil) \land$
$(\forall a, l \cdot l \in L \land a \in A \Rightarrow fr(cons(a \mapsto l)) = f(cons(a \mapsto l) \mapsto fr(l)))$

$FrLSB(fs \mapsto f0 \mapsto f)$
$= FrSB1(\{l, a, s \cdot l \in L \land a \in A \land s \in S | ((l \mapsto fs(s)) \mapsto (cons(a \mapsto l) \mapsto s)\} \mapsto f0 \mapsto f)$ then

$\forall fs, f0, f, fr \cdot fs \mapsto f0 \mapsto f \mapsto fr \in FrLSB$
$\Rightarrow fr \in L \to B \land fr(nil \mapsto s) = f0(nil \mapsto s) \land$
$(\forall a, l \cdot l \in L \land a \in A \Rightarrow fr(cons(a \mapsto l) \mapsto s) = f(cons(a \mapsto l) \mapsto s \mapsto fr(l \mapsto fs(s))))$

For example, ms defines a function from A to $\mathbb{N}$ that yields the number of occurrences of a value in the list (ms standing for multi set).

$ms(nil) = A \times \{0\}$ and

$\forall a, l \cdot a \in A \wedge l \in L \Rightarrow ms(cons(a \mapsto l)) = ms(l) \Leftarrow \{a \mapsto ms(l) + 1\}$

then $val = (\lambda l \cdot l \in L | dom(ms(l) \rhd \{0\}))$. The function val gives the set of value in the list. We can prove that $val(nil) = \varnothing$ and $\forall a, l \cdot a \in A \wedge l \in L \Rightarrow val(cons(a \mapsto l)) = val(l) \cup \{a\}$

if ord is a total order on A we can give the insertion function $insert$:

$insert(nil \mapsto v) = cons(v \mapsto nil)$

$insert(cons(a \mapsto l) \mapsto v) = cons(v \mapsto cons(a \mapsto l))$ if $v \mapsto a \in ord$

$insert(cons(a \mapsto l) \mapsto v) = cons(a \mapsto insert(l \mapsto v))$ if $v \mapsto a \notin ord$

and an insertion sorting function on lists:

$sort(nil) = nil$ and

$\forall a, l \cdot a \in A \wedge l \in L \Rightarrow sort(cons(a \mapsto l) = insert(sort(l) \mapsto a)$

5 Ordered lists

An empty list is ordered by definition. The list $cons(a \mapsto l)$ is ordered if l is ordered and all values in l are greater than a (relative to the total order). The fixpoint operator is instantiated with $S := L$. Let $ordL$ denotes the set of ordered lists, we have: $ordL = \text{fix}(\lambda lo \cdot lo \subseteq \mathbb{P}(L) | \{nil\} \cup cons[a \mapsto l | a \in A \wedge l \in lo \wedge (\forall v \cdot v \in val(lo) \Rightarrow a \mapsto v \in ord)\}])$ then we can prove the following theorem:

$ordL = \{nil\} \cup \{cons(a \mapsto l) | a \in A \wedge l \in ordL \wedge (\forall v \cdot v \in val(l) \Rightarrow a \mapsto v \in ord)\}])$.

Now we can prove that $sort$ is a sort function. The result is an ordered list and contains the same values with the same occurrences.

$\forall l \cdot l \in L \Rightarrow sort(l) \in ordL \wedge ms(l) = ms(sort(l))$

6 Conclusion

The fixpoint operator (implemented as a function in Rodin) is used multiple times to define lists, recursive functions and ordered lists. To initiate the typing process for lists, a carrier set is required. Additional well-founded relations are defined for other recursive sorting functions on list such as merge sort, quicksort and selection sort. Notably, we have consistently used the respective well-founded relation to prove the correctness of each sorting functions.

We have also applied analogous constructions to define binary trees, an inductive subset for the Goodstein sequence, and specific trees for Kruskal theorem.

After the Rodin worship an archive will be available on my website.

References

1. J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010
2. D. Cansell, J.-R. Abrial: *Examples of using the Instantiation Plug-in"*, Rodin Workshop 2021
3. D. Cansell: *Mathematics and software development using instantiation* EBRP 2025. `http://dominique.cansell.free.fr/JRAInstantiation.pdf`
4. G. Verdier and L. Voisin *Context instantiation plug-in: a new approach to genericity in Rodin.*, Rodin Workshop 2021
5. EBRP *Enhancing EventB and Rodin.* `https://irit.fr/EBRP`
6. *Rodin Platform.* `http://wiki.event-b.org`