Simulating Induction-Recursion for Partial Algorithms

Dominique Larchey-Wendling\(^1\) and Jean-François Monin\(^2\)

\(^1\) Université de Lorraine, CNRS, LORIA
dominique.larchey-wendling@loria.fr

\(^2\) Université Grenoble Alpes, CNRS, Grenoble INP, VERIMAG
jean-francois.monin@univ-grenoble-alpes.fr

Abstract

We describe a generic method to implement and extract partial recursive algorithms in Coq in a purely constructive way, using L. Paulson’s if-then-else normalisation as a running example.

Implementing complicated recursive schemes in a Type Theory such as Coq is a challenging task. A landmark result is the Bove & Capretta approach [BC05] based on accessibility predicates, and in case of nested recursion, simultaneous Inductive-Recursive (IR) definitions of the domain/function [Dyb00]. Limitations to this approach are discussed in e.g. [Set06, BKS16]. We claim that the use of (1) IR, which is still absent from Coq, and (2) an informative predicate (of sort Set or Type) for the domain, preventing its erasing at extraction time, can be circumvented through a suitable bar inductive predicate.

The recursive definition of Fig. 1 can be extracted as is from the Coq term that implements \(\text{nm}\).

We start with the inductive definition of the graph \(\mathcal{G} : \Omega \rightarrow \Omega \rightarrow \text{Prop}\) of \(\text{nm}\) (Fig. 2) and we show its functionality.\(^1\) Then we define the domain/termination predicate \(\mathcal{D} : \Omega \rightarrow \text{Prop}\) as a bar inductive predicate with the three rules of Fig. 3.

We illustrate our technique on L. Paulson’s algorithm for if-then-else normalisation [Gie97, BC05] displayed in Fig. 1. For concise statements, we use \(\omega\) to denote the ternary constructor for if\(_{\text{then-else}}\) expressions, and \(\alpha\) as the nullary constructor for atoms. As witnessed in the third match rule \(\omega(\alpha(a,b,c), y, z) \Rightarrow \text{nm}(\alpha(\alpha(a,\text{nm}(\omega(b,y,z)), \text{nm}(\alpha(c,y,z))))\)), we proceed purely constructively with the existing Coq system and

\(\text{nm} \colon \alpha \rightarrow \Omega + \Omega \rightarrow \Omega\)

\(\text{let nrec e = match e with}
\ |
\ |
\ \alpha \Rightarrow \alpha
\ |
\ \omega(\alpha,y,z) \Rightarrow \omega(\alpha,\text{nm}(y,\text{nm}(z)))\)

We proceed purely constructively without any extension to the existing Coq system and

\(\text{D} y \rightarrow \text{D} z\)

\(\text{D} \alpha \rightarrow (\omega \alpha y z)\)

\(\forall n, n_e, G (\alpha b y z) n_e \rightarrow G (\alpha c y z) n_e \rightarrow D (\alpha a n_e)\)

\(\text{D} (\omega (\alpha a b c) y z)\)

Finally, we define \(\text{nm} e D_e := \pi_1(\text{nm}_{\text{rec}} e D_e)\) and get \(\text{nm}_{\text{spec}} e D_e : G e (\text{nm} e D_e)\) using the second projection \(\pi_2\). Extraction of OCaml code from \(\text{nm}\) outputs exactly the algorithm of Fig. 1, illustrating the purely logical (Prop) nature of \(D_e\).

In order to reason on \(\mathcal{D} / \text{nm}\) we show that they satisfy the IR specification given in Fig. 4: the constructors of \(\mathcal{D}\) are sufficient to establish the simulated constructors \(d_{\text{nm}}[012]\), while \(\text{nm}_{\text{spec}}\) allows us to derive the fixpoint equations of \(\text{nm}\).

\(^1\) i.e. \(g_{\text{nm}}_{\text{fun}} : \forall e n_1 n_2, G e n_1 \rightarrow G e n_2 \rightarrow n_1 = n_2\).
ing \texttt{g\_nm\_fun}, we get proof-irrelevance of \texttt{nm}.\footnote{\texttt{nm\_pirr} : \texttt{\forall v e D_1 \exists e e D_1 = \texttt{nm e D_2}.}}

\begin{verbatim}
Inductive \texttt{\Omega} \texttt{Set} := \alpha : \texttt{\Omega} | \omega : \texttt{\Omega \rightarrow \Omega} \rightarrow \texttt{\Omega \rightarrow \Omega}.
Inductive \texttt{D} : \texttt{\Omega \rightarrow Prop} :=
| \texttt{d\_nm\_0} : \texttt{\alpha} |
| \texttt{d\_nm\_1 y c} : \texttt{D y \rightarrow D (\alpha a y c)} |
| \texttt{d\_nm\_2 a b c y z D_0 D_1} : \texttt{D (\alpha a (\texttt{nm (\alpha b y z) D_0})) (\texttt{nm (\alpha c y z) D_1})} \rightarrow \texttt{D (\alpha (\alpha a b c y z))}\end{verbatim}

\begin{verbatim}
with \texttt{Fixpoint \_nm e (D_2 : \texttt{\texttt{\Omega} e}) : \texttt{\Omega} := \texttt{match D_2 with}
| \texttt{d\_nm\_0} -> \texttt{\alpha} |
| \texttt{d\_nm\_1 y c D_1 D_3} -> \omega \alpha (\texttt{nm y D_1}) (\texttt{nm z D_1}) |
| \texttt{d\_nm\_2 a b c y z D_0 D_1 D_3} \rightarrow \texttt{\texttt{nm (\alpha a (\texttt{nm b y z) D_0}) (\texttt{nm (\alpha c y z) D_1}) D_3} | D_1 end.}

Figure 4: IR spec. of \texttt{D} : \texttt{\Omega \rightarrow Prop} and \texttt{nm} : \texttt{\forall e \texttt{D} e \rightarrow \texttt{\Omega} .}

We show a dependent induction principle for \texttt{D} (see Fig. 5). The term \texttt{\_nm\_ind} states that any dependent property \texttt{P} : \texttt{\forall e D \rightarrow Prop} contains \texttt{D} as soon as it is closed under the simulated constructors \texttt{d\_nm [012]} of \texttt{D}. The assumption \texttt{\forall e D_1 D_2, P e D_1 \rightarrow P e D_2} restricts the principle to \texttt{proof-irrelevant} properties about the dependent pair \texttt{(e, D_e)}. This is exactly what we need to establish properties of \texttt{nm}. Then we can show partial correctness and termination as in \cite{Gie97} – in this example, \texttt{nm} happens to always terminate on a normal form of its input. In a more relational approach, these properties can alternatively be proved using \texttt{nm\_spec} and induction on \texttt{G x nm}.\footnote{Our Coq code is available under a Free Software license [LWM18]. We have successfully implemented other algorithms using the same technique: F91, unification, depth first search as in [Kra10], quicksort, iterations until 0, partial list map as in \cite{BKS16}, Huet\&Hullot’s list reversal [Gie97], etc. The method is not constrained by nested/mutual induction, partiality or dependent types. On the other hand, spotting recursive sub-calls implies the explicit knowledge of all the algorithms that make such calls, a limitation that typically applies to higher order recursive schemes such as e.g. substitutions under binders. Besides growing our bestiary of examples, we aim at formally defining a class of schemes for which our method is applicable, and more practically propose some automation like what is done in Equations [Soz10].}

\begin{verbatim}
Theorem \texttt{\_nm\_ind (\forall P : \texttt{\forall e D \rightarrow Prop}) :}
| \texttt{\forall e D_1 D_2, P e D_1 \rightarrow P e D_2)} \rightarrow (P \_d\_nm\_0)
| \texttt{\forall \alpha a y c D_1 D_3) \rightarrow (P \_d\_nm\_1 y c D_1 D_3)
| \texttt{\forall \alpha a (\texttt{nm (\alpha b y z) D_0} D_0) (\texttt{nm (\alpha c y z) D_1 D_3) \rightarrow (P \_d\_nm\_2 a b c y z D_0 D_1 D_3)}]
| \texttt{\forall e D_1, P e D_1)}

Figure 5: Dependent induction principle for \texttt{D} : \texttt{\Omega \rightarrow Prop}.

Though our approach is inspired by IR definitions, in contrast with previous work, e.g. \cite{Bov09}, the corresponding principles are established \texttt{independently} of any consideration on the semantics or termination of the target function (\texttt{nm}), i.e. without proving any properties of \texttt{D / nm} a priori. This postpones the study of termination after both \texttt{D} and \texttt{nm} are defined together with constructors and elimination scheme, fixpoint equations and proof-irrelevance. Moreover, our domain/termination predicate \texttt{D} is \texttt{non-informative}, i.e. it does not carry any computational content. Thus the code obtained by extraction is exactly as intended.

References

\begin{itemize}
\end{itemize}