Synthesis of Programs in Abstract Data Types*

Alessandro Avellone   Mauro Ferrari   Pierangelo Miglioli
Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano
via Comelico 39, 20135 Milano–Italy
{avellone,ferram,miglioli}@dsi.unimi.it

1 Introduction

Starting from the pioneering work of Constable [4], Constructive Program Synthesis has been investigated by many authors. A non exhaustive list is [2, 8, 10, 12, 14]. The general idea is that a constructive proof of a specification S implicitly contains an algorithm to solve S.

A popular approach to Constructive Program Synthesis is known as *proofs as programs* (see, e.g., [2, 10, 11]). It is based on the fact that constructive proofs can be (almost) directly interpreted as executable programs, using for example, the Curry-Howard isomorphism, or other kinds of operational interpretations, like the one in [10].

In this abstract we follow a different, less popular approach, that we call *program–extraction*. Here, the relevant computational content of a constructive proof is extracted in the form of a program of a target programming language. We have chosen an imperative, Pascal-like programming language.

In program–extraction, we distinguish two kinds of proof-patterns: *active patterns*, that implicitly contain pieces of the program to be extracted, and *correctness patterns*, that prove correctness-properties, but are not relevant for extracting the program. Active and correctness patterns are recognized while analyzing a constructive proof, in the attempt of extracting a program. The constructive content of an active pattern is then immediately translated into a piece of a program by appropriate *synthesis rules*.

Correctness–patterns may be classical proofs, i.e. program–extraction needs weaker constructiveness assumptions, with respect to the *proofs as programs* approach. This is a strong point, in favour of the former.

On the other hand, in the *proofs as programs* approach, every constructive proof has a computational meaning, while in program-extraction there are proofs from which no program can be obtained. This seems to be a weak point, but the situation is not so bad.

We have proven that, if there is an Hoare-like total correctness proof for a program P, then there is a proof Π from which P can be extracted. So, the problem is to select the right proof-patterns. This problem cannot be avoided even in the *proofs as programs* approach, since many constructive proofs correspond to bad, naive programs: one has to select those specifications and proof–patterns, that correspond to reasonable programs.

The paper is so organized. Next section

*Extended Abstracts of LOPSTR'98, Eighth International Workshop on Logic-based Program Synthesis and Transformation, 15-19 June 1998, Manchester, UK. Technical Report Series, Department of Computer Science, University of Manchester, ISSN 1361-6161. Report number UMCS-98-6-1. http://www.cs.man.ac.uk/cstechrep/titles98.html

briefly introduces the specification languages and the underlying logical notions. Then, in Section 2 we describe the active proof patterns, based on the notion of *dischargeable set*, while in Section 3 we explain the synthesis rules. Finally, a few conclusion remarks round off the paper.

2 The Specification Languages

Specifications are written in the context of a first order axiomatization $\mathcal{A} = \langle \Sigma, \mathbf{T} \rangle$ of an *abstract data type* (ADT), where:

- Σ is a (many sorted) first order signature,
- $\mathbf{T}$ is a recursive set of Σ-axioms, such that there is a reachable isoinitial model $\mathcal{I}_\Sigma$ of $\mathbf{T}$.

Reachability is known in literature also as "no junk" [6]. It means that every element of the domain is representable by a ground term. Isoinitial models are similar to the more popular initial models. $\mathcal{I}$ is an isoinitial model of $\mathbf{T}$ iff, for every model $\mathcal{M}$ of $\mathbf{T}$, there is an unique isomorphic embedding $\iota : \mathcal{I} \to \mathcal{M}$. For an initial model, homomorphisms are used in the place of isomorphic embeddings. Isoinitial models have been introduced in [3] and have been used in specification frameworks by [7]. Here we use isoinitial models because they seem more suitable to a constructive approach (see also [9]). In particular, we are interested in the following properties.

Definition 1 *A first order theory* $\mathbf{T}$ *is semiconstructive in a logical systems* $\mathbf{L}$ *iff :*

- $\mathbf{T} + L$ *is classically consistent, i.e., if* $\mathbf{T} \vdash_{\mathbf{CL}} \bot$*, then* $\mathbf{T} \vdash_{\mathbf{L}} \bot$*;* [1]
- *for every closed formula* $A \vee B$*,* $\mathbf{T} \vdash_{\mathbf{L}} A \vee B$ *implies* $\mathbf{T} \vdash_{\mathbf{CL}} A$ *or* $\mathbf{T} \vdash_{\mathbf{CL}} B$*;*
- *for every closed formula* $\exists x A(x)$*,* $\mathbf{T} \vdash_{\mathbf{L}} \exists x A(x)$ *implies* $\mathbf{T} \vdash_{\mathbf{CL}} A(t)$*, for at least one ground term* t*.*

Semiconstructive systems have been introduced and discussed in [9]. Here we are interested in the following property.

Theorem 2 *Let* $\mathbf{T}$ *be a* Σ*-theory with at least one reachable model and let us suppose that* $\mathbf{T}$ *is semiconstructive in* $\mathbf{L}$*. Then* $\mathbf{T}$ *has a reachable isoinitial model* $\mathcal{I}$ *iff* $\mathbf{T} \cup \{\forall x(A(x) \vee \neg A(x)) \mid A$ *atomic*$\}$ *is semiconstructive in* $\mathbf{L}$*.* □

The above result is particularly important, in our approach, by the role that formulae of the kind $A \vee \neg A$ will play in dischargeable sets.

Now we are ready to introduce specifications.

Given any ADT-axiomatization $\mathcal{A} = \langle \Sigma, \mathbf{T} \rangle$ with isoinitial model $\mathcal{I}$, the logical definition language of $\mathbf{T}$, we call motivation language of $\mathbf{T}$, is defined as follows:

Motivation:

- assignmot:=[$\exists$ VAR* (WFF)] / $\ll \exists$ VAR* (WFF)$\gg$
- testmot:=[WFF]

where VAR and WFF are, respectively, any variable and any Σ-formula. We interpret the wffs in the isoinitial model $\mathcal{I}$, in the usual classical way, while assignmots and testmots have the following meaning:

– An assignmot $[\exists z^* \psi(\underline{x}, z)]$ expresses two distinct tasks:

1. Compute, for every n-tuple $\underline{a}$ of the appropriate sort in $\mathcal{I}$, the truth value of the predicate "$\exists z \psi(\underline{x}, z)$", i.e., establish whether either $\mathcal{I} \models \exists z \psi(\underline{x}, z)$ or $\mathcal{I} \models \neg \exists z \psi(\underline{x}, z)$ (assigning the values $\underline{a}$ to the variables $\underline{x}$ respectively).
2. For every $\underline{a}$ of the appropriate sort in $\mathcal{I}$ such that $\mathcal{I} \models \exists z \psi(\underline{x}, z)$, compute a value b such that $\mathcal{I} \models \psi(\underline{x}, z)$ (assigning the values $\underline{a}, b$ to the variables $\underline{x}, z$ respectively).

– An assignmot $\ll \exists z^* \psi(\underline{x}, z) \gg$ expresses the task of computing a value b such that $\mathcal{I} \models \psi(\underline{x}, z)$ for every n-tuple $\underline{a}$ of the appropriate sort in $\mathcal{I}$ (assigning the values $\underline{a}, b$ to the variables $\underline{x}, z$ respectively).

– A testmot $[\psi(\underline{x})]$ expresses the task of computing the truth-value of the predicate "$\psi(\underline{x})$" for every n-tuple $\underline{a}$ of the appropriate sort in $\mathcal{I}$ (i.e., of stating wheter or not $\mathcal{I} \models \psi(\underline{x})$ assigning the values $\underline{a}$ to the variables $\underline{x}$).

The only fact that a motivation is written does not guarantee, of course, the possibility of

[1] $\vdash_{\mathbf{L}}$ means provability in $\mathbf{L}$, $\vdash_{\mathbf{CL}}$ means provability in classical logic.

accomplishing the implied task. So, an analisys of the soundness of a motivation is needed. We carry out such an analysis by means of the deductive apparatus of $\mathbf{T}+\mathbf{L}$. We say that:

1. *An assignmot* $[\exists z^{*}\psi]$ *is sound* iff $\mathbf{T}\vdash_{\mathbf{L}}\exists z\psi \vee \neg\exists z\psi$;
2. *An assignmot* $\ll \exists z^{*}\psi \gg$ *is sound* iff $\mathbf{T}\vdash_{\mathbf{L}}\exists z\psi$;
3. *A testmot* $[\psi]$ *is sound* iff $\mathbf{T}\vdash_{\mathbf{L}}\psi \vee \neg\psi$.

If $\mathbf{T}+\mathbf{L}$ is semiconstructive, then it is guarantied that a sound motivation can be solved in its classical extention $\mathbf{T}+\mathbf{CL}$. The problem is: can we find a proof in $\mathbf{T}+\mathbf{L}$ from which we can extract a "good" algorithm?

As we will see in the next sections we can solve the above problem if the proof follows special patterns, "active patterns", which are based on dischargeable sets.

3 Active Proof Patterns

As we have already said, our active proof patterns are based on the construction of particular sets of wffs, we call *$\phi\psi$-dischargeable sets*. The idea is to build a set $\mathtt{I}$ of sets of wffs (the $\phi\psi$-dischargeable set) to which we can associate a proof in $\mathbf{T}+\mathbf{L}$ of the wff $\psi \vee \neg\psi$ (or of the wff $\exists z\psi(z)$) where any undischarged assumption either belongs to some set in $\mathtt{I}$ or is ϕ. Moreover, starting from the above formal proof and assuming other properties on $\mathtt{I}$, one can obtain a proof in $\mathbf{T}+\mathbf{L}$ of the wff $\psi \vee \neg\psi$ (or of the wff $\exists z\psi(z)$) where the only undischarged assumption is the wff ϕ.

To better understand how our proof method is related to formal proofs, let us consider a simple example in the context of first order intuitionistic Peano arithmetic (**IPA** for short) presented in a natural deduction style calculus.

Let $\psi(x,y,z) \equiv \exists y_1, y_2\ (x{+}y_1{=}z \wedge z{+}y_2 = y \wedge \neg y_1 = 0)$ ("$\psi(x,y,z)$" represents the predicate "$x{<}z{\leq}y$"), $A \equiv \exists y_2(x{+}y_2{=}y)$, $B \equiv x{=}y$ and $C' \equiv \exists y_1, y_2\ (x{+}y_1{=}s(x) \wedge s(x){+}y_2{=}y \wedge \neg y_1{=}0)$.

If there are a proof Π of $A \vee \neg A$ and a proof Π' of $B \vee \neg B$, then one can construct the following proof ($\overline{\overline{\Pi}}$) of $\exists z\psi \vee \neg\exists z\psi$, where $C \equiv \exists z\psi(x,y,z)$):

$$
\dfrac{\Pi\ \dfrac{\begin{matrix}\{\neg A\}\\ \vdots\ \pi_1\\ \neg C\end{matrix}}{C \vee \neg C}\,\vee I
\qquad
\dfrac{\Pi'\ \dfrac{\begin{matrix}\{A\},\{\neg B\}\\ \vdots\ \pi_2\\ C'\end{matrix}}{C}\,\exists I}{C \vee \neg C}\,\vee I
\quad
\dfrac{\begin{matrix}\{B\}\\ \vdots\ \pi_3\\ \neg C\end{matrix}}{C \vee \neg C}\,\vee I
}{\ldots}
$$

$$
\dfrac{\Pi\quad C \vee \neg C \qquad \dfrac{\Pi'\quad C \vee \neg C \qquad C \vee \neg C}{C \vee \neg C}\,\vee E}{\exists z\psi(x,y,z) \vee \neg\exists z\psi(x,y,z)}\,\vee E
$$

This proof–tree can be obtained by analyzing the $\exists z\psi(x,y,z)\vee\neg\exists z\psi(x,y,z)$-dischargeability (see Def. 3) of the assumptions of π_1, π_2, π_3, i.e. of the set $\{\{\neg A\},\{A,B\},\{A\neg B\}\}$. Dischargeability does not depend on the specific content of Π, Π', π_1, π_2, π_3, but only on the possibility of applying the inference rules that have been shown in $\overline{\overline{\Pi}}$. The dischargeability analysis provides the wanted proof by reconstructing the outermost active patterns of the proof, namely the $\vee$-introduction and elimination and the $\exists$-introduction shown in the above proof-tree. At the same time, it allows to translate such active patterns into program constructs.

These considerations should explain why our synthesis method only translate the rules of $\exists$-introduction (assignments), of $\vee$-introduction (assignment to the `tv` variable) and of $\vee$-elimination (**if-then-else**).

Here, we will not give all the definitions related to $\phi\psi$-dis sets (see, [1] for a complete definition and discussion on dischargeable sets); we only give the main definitions and results.

Thus, let $\mathtt{I} = \{\Gamma_1, \ldots, \Gamma_n\}$ be any set of classically consistent sets of wffs[2] not containing the empty set. First of all, by Γ *occurs conjugated in* $\mathtt{I}$ *with respect to a wff* A we mean that there is a $\Delta \in \mathtt{I}$ such that $A \in \Gamma$ and $\neg A \in \Delta$; moreover, either $\Gamma \setminus \{A\} \subseteq \Delta \setminus \{\neg A\}$ or $\Delta \setminus \{\neg A\} \subseteq \Gamma \setminus \{A\}$. Now, let ϕ and ψ be two wffs; we say that $\mathtt{I}$ is a *$\phi\psi$-dis set* iff one of the following conditions is satisfied:

1. $\mathtt{I}$ is empty;
2. there are a $\Gamma_i \in \mathtt{I}$ $(1 \leq i \leq n)$ and a wff A such that Γ_i occurs conjugated in $\mathtt{I}$ with respect to A, $\phi\vdash_{\mathbf{T+L}} A \vee \neg A$, and there is a set $\mathtt{I}_A^{\Gamma_i}$ which is $\phi\psi$-dis;

[2]One can give less restrictive condition on this set of formulae rather than classical consistency

3. $\mathtt{I} = \{\Gamma\}$ $(n = 1)$ and there is a wff $A(x) \in \Gamma$ such that, for all $B \in \Gamma$, if $B \not\equiv A$ then x is not a free variable of the wffs B, ϕ and ψ, $\phi \vdash_{\mathbf{T+L}} \exists z\ A(z)$, and the set $\mathtt{I}^* = \{\Gamma \setminus \{A\}\}$ is $\phi\psi$-dis.

Definition 3 *If ϕ and ψ are two wffs such that $\phi \not\equiv \exists v\ \phi'$ for every variable v and every wff ϕ', then a $\phi\psi$-dis set $\mathtt{I} = \{\Gamma_1, \ldots, \Gamma_n\}$ is said to be* ϕ-dis w.r.t $\psi \vee \neg\psi$ *iff, for every $\Gamma_i \in \mathtt{I}$ $(1 \leq i \leq n)$, either $\phi, \Gamma_i \vdash_{\mathbf{T+L}} \psi$, or $\phi, \Gamma_i \vdash_{\mathbf{T+L}} \neg\psi$. If $\phi \equiv \exists v\ \phi'$ and v does not occur free in ψ, then $\mathtt{I}$ is ϕ-dis w.r.t. $\psi \vee \neg\psi$ iff $\mathtt{I}$ is $\phi'\psi$-dis w.r.t. $\psi \vee \neg\psi$.*

The distinction in Definition 3 is related to the synthesis rules.

Proposition 4 *If $\mathtt{I}$ is ϕ-dis w.r.t. $\psi \vee \neg\psi$, then $\phi \vdash_{\mathbf{T+L}} \psi \vee \neg\psi$.*

The proof of the above proposition can be accomplished by induction on the number of sets contained in the dischargeable set $\mathtt{I}$. An analysis of this proof shows that $\phi \vdash_{\mathbf{T+L}} \psi \vee \neg\psi$ can be proved by patterns of the kind $\phi, \Gamma \vdash_{\mathbf{T+L}} \psi$ or $\phi, \Gamma \vdash_{\mathbf{T+L}} \neg\psi$ where Γ is an appropriate set of formulae obtained by the sets contained in $\mathtt{I}$. These patterns will be the input parameters of our synthesis rule.

The $\phi\psi$-dis sets underly proofs in $\mathbf{T} + \mathbf{L}$ of facts such as $\phi \vdash_{\mathbf{T+L}} \psi \vee \neg\psi$; in particular, when ϕ is the empty wff $\emptyset$, a set $\emptyset$-dis w.r.t. $\exists z\psi \vee \neg\exists z\psi$ (or w.r.t. $\psi \vee \neg\psi$ or w.r.t. $\exists z\psi$) underlies the soundness proof of an assignmot $[\exists z^*\psi]$ (respectively the soundness proof of a testmot $[\psi]$ or the soundness proof of an assignmot $\ll \exists z^*\psi \gg$).

We also have analyzed other principles which can be used to prove the soundness of the assignmots (of the testmots). Our aim is to single out some principles giving rise to "good synthesis rules".

As one should expect in ADT-theories, in the non trivial cases a proof of $\psi \vee \neg\psi$ is made using some inductive principles. Hence we need to investigate the relation between $\phi\psi$-dis sets and induction principles.

Induction Principle
As an example, we consider first order induction schema in the frame of **IPA**. Again, the induction proofs are based on the construction of suitable dischargeable sets which depends on the motivation to be proved. Essentially, the proof (by induction, for instance, on the variable x_1) is carried out in three steps: (**Basis**) a proof that $\vdash_{\mathbf{IPA}} \psi(\mathsf{0}) \vee \neg\psi(\mathsf{0},)$ is provided by constructing a $\emptyset$-dis set respect to $\psi(\mathsf{0}) \vee \neg\psi(\mathsf{0})$. (**Step+**) a proof that $\psi(j) \vdash_{\mathbf{IPA}} \psi(\mathsf{s}(j)) \vee \neg\psi(\mathsf{s}(j))$ is given, by constructing a set $\psi(j)$-dis with respect to $\psi(\mathsf{s}(j)) \vee \neg\psi(\mathsf{s}(j))$. (**Step-**) a proof that $\neg\psi(j) \vdash_{\mathbf{IPA}} \psi(\mathsf{s}(j)) \vee \neg\psi(\mathsf{s}(j))$ is given, by constructing a set $\neg\psi(j)$-dis with respect to $\psi(\mathsf{s}(j)) \vee \neg\psi(\mathsf{s}(j))$.

Descending Chain Principle
We analyze the case where one has to prove the soundness of an assignmot $[\exists z^*\psi(x,z)]$ (or the soundness of a testmot $[\psi(x,z)]$) using the Descending Chain Principle (DCH) formulated as follows:

$$\dfrac{\begin{array}{c} \\ \vdots\ \pi_1 \\ \exists \underline{z} A(z,v) \end{array} \quad \begin{array}{c} \ldots A(y,v) \\ \vdots\ \pi_2 \\ \exists z((A(z,v) \wedge z \prec y) \vee B(z,w)) \end{array}}{\exists z\ B(z,w)}\ {\scriptstyle DCH}$$

where v and w are the only free variables occurring in A and B respectively, and y is the *eigenvariable* of the rule and does not occur free in B and in the other undischarged assumptions of the proof π_2. Moreover, we assume that the relation symbol $\prec$ is interpreted as a well founded order relation between elements of the appropriate sorts in the isoinitial model $\mathcal{I}$ of $\mathbf{T}$. $A(z,v)$ is called the *DCH-formula*. One has to assume that $A(z,v) \equiv \exists t_1 \ldots \exists t_k\ A'(z,v,\underline{t})$, where $A' \not\equiv \exists r\ A''(r)$ for any variable r and every wff A''. Again, the proof is carried out by considering separately the two premises of the rule.

4 Synthesis rules

We will use a PASCAL like programming language (PL) containing the usual structures **if-then-else**, **do-while**, **while-do-end** and the composition ";" to present the synthesis rules. So, it is possible to directly apply the well known Hoare's rules to inspect the correctness of the synthesis rules. Moreover, we will need a special variable "`tv`" which is intended to store the truth-value of the testmots (assignmots) associated with the program.

We only explain the synthesis rules corresponding to dischargeable sets (called Ξ-rules) and to induction rule for assignmots, we call *ordinary synthesis rules*; in section 5 we will give an example involving active patterns for the descending chain principle. We also point out that a special expansion rule to complete the rules for $\phi\psi$-dis sets must be given. Indeed, in proving the soundness of a motivation we proceed top-down, i.e., we construct $\phi\psi$-dis sets containing wffs A for which $\Delta\vdash_{\mathtt{T+L}} A \vee \neg\, A$ or $\Delta\vdash_{\mathtt{T+L}} \exists x A(x)$ (assuming Δ to be a finite and possibly empty set) must be, in turn, proved. For such a wff A we assume, in the top level, $\Delta\vdash_{\mathtt{T+L}} A \vee \neg A$ or $\Delta\vdash_{\mathtt{T+L}} \exists x A(x)$ and we construct a program containing the high level test $[A]$ or the assignment $\ll \exists z^* A(z) \gg$ which depends on Δ; then we take $[A]$ as a new testmot or we take $\ll \exists z^* A(z) \gg$ as a new assignmot and we construct a program from the proof of its soundness.

Now, before explaining the ordinary synthesis rules some comments are in order. For each application of an ordinary synthesis rule we construct a program. Such a program will be given enclosed between brackets, while a set of wffs on the left and a wff on the right will specify its meaning. The wffs on the left and the wff on the right of the bracketed program are respectively the set of the input assertions and the output assertion of the program in the sense of Floyd and Hoare. When a Ξ-rule is applied to an assignmot $[\exists z^* \psi(x,z)]$ where $\psi(x,z) \equiv \exists \underline{y} \psi(x,z,\underline{y})$, the variables $\underline{y}, z$ are taken as input variables which can be modified by the program; therefore, the assertions on the left contain free the variables $x, \underline{y}, z$, while the assertion on the right contains free only the variables x, z and `tv`.

4.1 Ordinary rules for ϕ-dischargeable sets

Let `I` be a set ϕ-dis w.r.t $\exists z \psi(x,z) \vee \neg \exists z \psi(x,z)$ such that $\phi \equiv \exists y_1 \ldots \exists y_m \phi'(\underline{y})$ $(m \geq 0)$ and $\psi(x,z) \equiv \exists v_1 \ldots \exists v_n \psi(x,z,\underline{v})$ $(n \geq 0)$. The synthesis procedure is described in the following points:

1. Inspect for every $\Gamma \in$ `I` whether either
   (a) $\phi', \Gamma\vdash_{\mathtt{T+L}} \exists z \psi(x,z)$ or
   (b) $\phi', \Gamma\vdash_{\mathtt{T+L}} \neg\exists z \psi(x,z)$.
   If (a) is verified the input of the synthesis rule are the active patterns of $\phi', \Gamma\vdash_{\mathtt{T+L}} \psi(x,\underline{\tau},\tau)$ (from which (a) has been deduced); the rule is:

$$\frac{\phi', \Gamma\vdash_{\mathtt{T+L}} \psi(x,\underline{\tau},\tau)}{\phi', \Gamma\{v_1 := \tau_1; \ldots; v_n := \tau_n; z := \tau; \mathtt{tv} := 1\}_{\mathtt{Out}}}\ \Xi_1 R_1$$

   If b) is verified the input of the synthesis rule are the active patterns of $\phi', \Gamma\vdash_{\mathtt{T+L}} \neg\exists z \psi(x,z)$; hence we apply the rule

$$\frac{\phi', \Gamma\vdash_{\mathtt{T+L}} \neg\exists z \psi(\underline{x},z)}{\phi', \Gamma\{\mathtt{tv} := 0\}_{\mathtt{Out}}}\ \Xi_1 R_2$$

   where `Out` is the output assertion $(\mathtt{tv} = 1 \wedge \psi(\underline{x},z)) \vee (\mathtt{tv} = 0 \wedge \neg\exists z \psi(\underline{x},z))$.
2. When the previous point has been carried out for every $\Gamma \in$ `I`, apply repeatedly the discharging rules:

$$\frac{\phi', \Gamma_1, A\{P_1\}_{\mathtt{Out}} \qquad \phi', \Gamma_1', \neg A\{P_2\}_{\mathtt{Out}}}{\phi', \Gamma_1^*\{\textbf{if } A^* \textbf{ then } P_1 \textbf{ else } P_2\}_{\mathtt{Out}}}\ \Xi_1 R_3$$

   or

$$\frac{\phi', \Gamma^*, A(m)\{P\}_{\mathtt{Out}}}{\phi', \Gamma^*\{A^\star; P\}_{\mathtt{Out}}}\ \Xi_1 R_3'$$

   where:
   - $A^* = A$ if A is an "elementary" test of PL, or $A^* = [A]$ otherwise; $A^\star = \ll \exists z^* A(z) \gg$
   - Γ_1, Γ_1', Γ^*, and Γ_1^* are subsets of sets in `I`;
   - P_1 and P_2 are programs of PL constructed by previous applications of the synthesis rules.

Since `I` is a set ϕ-dis with respect to $\exists z \psi(\underline{x},z) \vee \neg\exists z \psi(\underline{x},z)$, there is a suitable sequence of applications of the above rules leading to $\phi'\{P\}$ `Out`, which is the desired result.

4.2 The induction rule

We consider the rule of induction in **IPA**. Let us suppose that there are (1) a set $\mathtt{I}_B$ which is $\emptyset$-dis with respect to $\exists z\psi(0, X, z) \vee \neg\exists z\psi(0, X, z)$ (X stands for $x_2, \ldots, x_n$), (2) a set $\mathtt{I}_+$ which is $\exists z\psi(j, X, z)$-dis with respect to $\exists z\psi(\mathtt{s}(j), X, z) \vee \neg\exists z\psi(\mathtt{s}(j), X, z)$, and (3) a set $\mathtt{I}_-$ which is $\neg\exists z\psi(j, X, z)$-dis with respect to $\exists z\psi(\mathtt{s}(j), X, z) \vee \neg\exists z\psi(\mathtt{s}(j), X, z)$; then a program is constructed according the following steps:

1. Construct by the rules $\Xi_1 R_1$, $\Xi_1 R_2$, and $\Xi_1 R_3$ the programs P_B, P_+, and P_- such that $\{P_B\}$ $\mathtt{Out}(0)$, $\psi(j, X, z)$ $\{P_+\}$ $\mathtt{Out}(\mathtt{s}(j))$, and $\neg\exists z\psi(j, X, z)\{P_-\}$ $\mathtt{Out}(\mathtt{s}(j))$; where $\mathtt{Out}(0)$ and $\mathtt{Out}(\mathtt{s}(j))$ have the obvious meaning.

2. Apply the following rule in order to arrange the three programs P_B, P_+, and P_-:

$$\frac{\begin{array}{l}\{P_B\}\ \mathtt{Out}(0);\\ \psi(j, X, z)\{P_+\}\ \mathtt{Out}(\mathtt{s}(j));\\ \neg\exists z\psi(j, z)\{P_-\}\ \mathtt{Out}(\mathtt{s}(j))\end{array}}{\begin{array}{l}\{P_B; j := 0;\\ \textbf{while } j \neq x_1 \textbf{ do}\\ \textbf{if } \mathtt{tv} = 1 \textbf{ then } P_+ \textbf{ else } P_-;\\ j := j{+}1\\ \textbf{end}\}\mathtt{Out}(x_1)\end{array}}\ Ind$$

5 An Example

Let us consider the formula $\exists z\psi(x, z)$, where $\psi(x, z) \equiv (z{*}z \leq x \wedge x{<}\mathtt{s}(z){*}\mathtt{s}(z))$. We want to construct a program (involving active patterns for the descending chain rule) corresponding to the assignmot $\ll \exists\ z^*\psi(x, z) \gg$, which expresses the intention of computing the value "z" of the integer square root of x.

1. $\mathcal{I}_B = \emptyset$ and

2. $\mathcal{I}_S = \{\{z = m{+}1, x{<}m{*}m\},$
$\{z = m{+}1, \neg x{<}m{*}m\}\}$

are the discharging sets needed in order to apply the descending chain rule, and $z{*}z = i \wedge x{<}i$ is the DCH–formula.

Thus, by **1** we immediately get the program:

$z := \mathtt{s}(x);$
$i := \mathtt{s}(x){*}\mathtt{s}(x);$
$\{\mathtt{Out}_1\}$

Considering $\mathcal{I}_S$, by the facts:

1. $z = m{+}1, x{<}m{*}m \vdash_{\mathbf{IPA}} m{*}m = m{*}m \wedge x{<}m{*}m \wedge m{<}z$ and

2. $z = m{+}1, \neg x{<}m{*}m, z{*}z = i \wedge x{<}i \vdash_{\mathbf{IPA}} m{*}m \leq x \wedge x{<}\mathtt{s}(m){*}\mathtt{s}(m)$,

using $\Xi_1 R_1$ we get the two programs

$\{z{*}z = i \wedge x{<}i,$ $z = m{+}1 \wedge x{<}m{*}m\}$	$\{z \cdot z = i \wedge x{<}i,$ $z = m{+}1 \wedge \neg x{<}m{*}m\}$
$z := m;$	$z := m;$
$i := m{*}m;$	$\mathtt{tv}_A := 2;$
$\mathtt{tv}_A := 1;$	$\mathtt{tv}_B := 1;$
$\mathtt{tv}_B := 2;$	$\{\mathtt{Out}_2\}$
$\{\mathtt{Out}_2\}$	

Hence, applying $\Xi_1 R3$ and $\Xi_1 R'3$ to the previous programs we obtain

$\{z{*}z = i \wedge x{<}i\}$
$(z = m{+}1)^\star;$
if $(x{<}m{*}m)$ **then**
$z := m;\ i := m{*}m;\ \mathtt{tv}_A := 1;\ \mathtt{tv}_B := 2;$
else $z := m;\ \mathtt{tv}_A := 2;\ \mathtt{tv}_B := 1;$
$\{\mathtt{Out}_2\}$

Finally, using the DCH–rule we get

$z := \mathtt{s}(x);\ i := \mathtt{s}(x){*}\mathtt{s}(x);$
do
$(z = m{+}1)^\star;$
if $(x{<}m{*}m)$ **then**
$z := m;\ i := m{*}m;\ \mathtt{tv}_A := 1;\ \mathtt{tv}_B := 2;$
else $z := m;\ \mathtt{tv}_A := 2;\ \mathtt{tv}_B := 1;$
while $\mathtt{tv}_A = 1$
$\{\mathtt{Out}_3\}$

Here $(z = m{+}1)^\star$ represents (in the case that the program language does not have as primitive function the predecessor function) the hight level program to compute the value of $m = z - 1$ under the assumption that $z{*}z = i \wedge x{<}i$.

6 Completeness

The aim of this section if to prove the following assertion in the frame of intuitionistic first order arithmetic: Given any testmot $[\psi(x_1, \ldots, x_n)]$ such that $\vdash_{\mathbf{IPA}} \psi(x_1, \ldots, x_n) \vee \neg\psi(x_1, \ldots, x_n)$, there is a program synthesized using dischargeable sets which computes the soundness of $[\psi(x_1,\ldots,x_n)]$.

Before proving the above assertion for any wff, we show how it holds for the representing wffs of primitive recursive functions; also, using a new synthesis rule, we will prove, *mutatis mutandis*, a corresponding assertion for wffs which represent, according to the well known Normal Form Theorem (Kleene 1938), partial recursive functions. Thus, we will start by providing a proof of the following assertion: let $f(x_1,\ldots,x_n)$ be any primitive recursive function and let $P(x_1,\ldots,x_n,w)$ be its representing wff; then $\vdash_{\mathbf{IPA}} P \vee \neg P$ is provable using dischargeable sets.

First of all, we can prove:

Theorem 5 *Let $f(x_1,\ldots,x_n)$ be any primitive recursive function and let $P(x_1,\ldots,x_n,w)$ be its representing wff; then $\vdash_{\mathbf{IPA}} \forall x_1 \ldots \forall x_n \exists! w P(x_1,\ldots,x_n,w)$ can be proved using dischargeable sets.*

Proof: The assertion can be proved by induction according the primitive recursive description of f using the strongly numeralwise representable property in **IPA** of primitive recursive functions (for a detailed definition we refer the reader to [1, 5]). □

As an immediate consequence of Theorem 5 one get: for every assignmot $\ll \exists z^* \psi(x_1,\ldots,x_n,z) \gg$, if $\psi(x_1,\ldots,x_n,z)$ is the representing wff of a primitive recursive function $f(x_1,\ldots,x_n)$, then $\ll \exists z^* \psi(x_1,\ldots,x_n,z) \gg$ can be synthesized using the above rules.

Now we prove.

Theorem 6 *Let $f(x_1,\ldots,x_n)$ be any primitive recursive function and let $P(x_1,\ldots,x_n,z)$ be its representing wff; then, $\vdash_{\mathbf{IPA}} \forall x_1 \ldots \forall x_n \forall z \; (P(x_1,\ldots,x_n,z) \vee \neg P(x_1,\ldots,x_n,z))$*

Proof: Again the assertion can be proved using the property of primitive recursive function to be strongly numeralwise representable in **IPA**. □

By inspecting the complete proof of Theorems 5 and 6, one can observe that the proof of $P(\underline{x},w) \vee \neg P(\underline{x},w)$ can be carried out by a formal induction where both the basis and the step cases can be proved using dischargeable sets. Hence

Corollary 7 *Let $f(x_1,\ldots,x_n)$ be a primitive recursive function and let $P(x_1,\ldots,x_n,w)$ be its representing wff; then $\vdash_{\mathbf{IPA}} P(x_1,\ldots,x_n,w) \vee \neg P(x_1,\ldots,x_n,w)$ can be proved using the dischargeable sets.* □

Corollary 8 *The synthesis method based on the dischargeable sets is complete with respect to the problem of deciding the graphs of primitive recursive functions, i.e., given any primitive recursive function $f(x_1,\ldots,x_n)$, for any $n+1$-tuple of natural numbers $x_1,\ldots,x_n,w$, one can build a program, using the synthesis method described below, which decides whether or not $P(x_1,\ldots,x_n,w)$ holds, where $P(x_1,\ldots,x_n,w)$ is the representing wff of $f(x_1,\ldots,x_n)$.* □

We can extend the previous result into a completeness result involving all the partial recursive functions with the help of new proof patterns involving Markov Principle. (see, e.g., [13]) and new synthesis rules; more details will be given in the full paper.

Moreover, we can also prove for our synthesis methods an *algorithmic completeness*, i.e., for every program P proved to be totally correct in a Hoare-like calculus, one can find a set of proof patterns giving P.

7 Conclusions

We have discussed an extractive approach to constructive program synthesis. This approach suggests to organize constructive proofs according to suitable patterns immediately translatable into programs. We have proved interesting completeness results.

The next step is to better understand the dependency of proof patterns on the target language, although they seem, indeed, to be rather independent, and to study proof-search strategies oriented to extractive program synthesis.

References

[1] A. Avellone. *The algorithmic content of constructive proofs: translating proofs into programs and interpreting proofs as programs.* PhD thesis, Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Italy, 1998.

[2] J. Bates and R. Constable. Proof as programs. *ACM Transaction on Programming Languages and Systems*, 7(1):113–136, 1985.

[3] A. Bertoni, G. Mauri, and P. Miglioli. On the power of model theory to specify abstract data types and to capture their recursiveness. *Fundamenta Informaticae*, VI(2):127–170, 1983.

[4] R. Constable. Constructive mathematics and automatic program writers. In *Proc. IFIP 1971 (Lublijana).* North–Holland, 1971.

[5] M. Ferrari. *Strongly Constructive Formal Systems.* PhD thesis, Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Italy, 1997.

[6] J. Goguen and J. Meseguer. Unifying functional, object-oriented and relational programming with logical semantics. In B. Shriver and editors P. Wegner, editors, *Research Directions in Object-Oriented Programming*, pages 417–477. MIT Press, 1987.

[7] K.K. Lau and M. Ornaghi. Forms of logic specifications: a preliminary study. In J. Gallagher, editor, *Proc. LOPSTR 96,* LNCS *1207*, pages 295–312, 1997. To appear.

[8] P. Miglioli, U. Moscato, and M. Ornaghi. Program specification and synthesis in constructive formal systems. In K.-K. Lau and T.P. Clement, editors, *Logic Program Synthesis and Transformation, Manchester 1991*, pages 13–26. Springer-Verlag, 1991. Workshops in Computing.

[9] P. Miglioli, U. Moscato, and M. Ornaghi. An improved refutation system for intuitionistic predicate logic. *Journal of Automated Reasoning*, 12:361–373, 1994.

[10] P. Miglioli and M. Ornaghi. A logically justified model of computation. *Fundamenta Informaticae*, IV,1,2:151–172,277–341, 1981.

[11] Chetan R. Murthy. Classical proofs as programs: How, what, and why. *Lecture Notes in Computer Science*, 613:71–??, 1992.

[12] Helmut Schwichtenberg. Proofs as programs. In P. Aczel, H. Simmons, and S. S. Wainer, editors, *Proof Theory. A selection of papers from the Leeds Proof Theory Programme 1990*, pages 81–113. Cambridge University Press, 1993.

[13] A. S. Troelstra. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis.* Lecture Notes in Mathematics 344. Springer–Verlag, 1973.

[14] A. A. Voronkov. On completeness of program synthesis systems. In E. Börger, G. Jäger, H. Kleine Büning, and M.M. Richter, editors, *Computer Science Logic*, pages 411–418. 5th Workshop, CSL '91, Berne, Switzerland, Springer–Verlag, October 1991 1991.