1

Extracting Logic Programs from Proofs that Use Extended Prolog Execution and Induction

LAURENT FRIBOURG [1]

In the constructive theory of types, several methods have been proposed to synthesize programs by extraction from proofs of existential statements of the form $\forall X \exists Y\ C(X,Y)$ (see, e.g., [CAea86, Hay86, Moh86]). The statement is seen as a specification of an output Y in terms of an input X. This branch has been less explored in the field of logic programming where synthesis methods are rather based on transformation techniques such as unfolding/folding (see, e.g., [CS77, Day87, Hog81, PP89, TS84]). A notable exception is [Tak87] which proposes to use the typed logical systems QJ in order to make proofs and extract programs in an intermediate code before compilation into Prolog. A similar idea is explained in [Bun88] using a logic based on Martin-Löf Type Theory [ML79], and is explored further in [BSW90, GWH91]. Also relevant is the work of [Eri84] and [Nar89] which propose proof-extraction methods based respectively on the natural programming calculus [HT79] and on the deductive-tableau system [MW80]. In this chapter, we propose instead to use an extension of the Prolog interpreter in order to make proofs, and we extract programs directly in Prolog. We thus exploit the fact that the computation mechanism of Prolog already implements a restricted form of the principle of proof extraction: a Prolog goal can

[1] This chapter is a revised and extended version of a paper presented under the same title at the Seventh International Conference on Logic Programming, Jerusalem, 1990.

Constructing Logic Programs, J.-M. Jacquet (editor)

be viewed as an existential statement of the form $\exists Y \ B(Y)$ where $B(Y)$ denotes a conjunction of atoms; the Prolog interpreter extracts an answer from the SLD-proof of the goal. What remains to be done in order to implement the full principle of proof extraction is to

i) enrich goals with universal variables X and with other connectives besides "$\wedge$".
ii) adapt the inference rule of SLD-resolution to this new class of goals, and add other inference rules, such as induction, for treating universal variables and the new connectives.
iii) provide a mechanism of program extraction in connection with the new inference system.

In order to achieve these points, we consider the proof system named "extended execution" [KS86, KF86, Kan88]. We will not apply this system to the general class of extended goals as defined in [KS86] but only to the class of "implicative goals" [Fri88]. Implicative goals are basically implicative formulas where the hypothesis and the conclusion are conjunctions of atoms. The use of implicative goals greatly simplifies the rules of extended execution as well as the proof extraction process because they contain just one implication connective and neither negation nor disjunction connectives. We will associate with every inference rule a rule of procedure construction. The constructed procedure relates a special atom (called *io*-atom, for "*i*nput-*o*utput") attached to the parent goal with the special atoms attached to the descendant goals. The synthesized program is the union of the procedures constructed during the proof.

The plan of the chapter is as follows. Section 1 introduces formally the notions of "implicative goal" and "*io*-atom", together with the relevant background. Section 1 describes the rules of extended execution and the associated rules of construction of the *io*-procedures. Section 1 states the correctness of the synthesis method. Section 1 presents a technique for removing superfluous arguments from the synthesized predicates. Finally, section 1 sums up the results of the chapter.

1.1

GOALS AND IO-ATOMS

We consider first-order formulas G of the form $\forall X \exists Y \ G^+ \Leftarrow G^-$, where X and Y are vectors of variables, and G^+ and G^- are conjunctions of atoms such that no variable of Y occurs in G^-. As in [KS86], we manipulate such formulas under a quantifier-free form. This quantifier-free form is obtained by replacing all the occurrences of any variable y of Y in G^+ by $?y$, and by dropping the quantifier part $\forall X \exists Y$. Under this new form, the formula G is called an *implicative goal*. It contains variables of the form $?y$ which are called the *undecided variables* of G while the other variables are called the *free variables* of G. The formula G will be also denoted $G^+(?Y) \Leftarrow G^-$ in order to make appear the vector $?Y$ of undecided variables. (We will also use $G^+(V, ?Y) \Leftarrow G^-(V)$ in order to make appear a subvector V of free variables.) The subformula G^- is the *hypothesis* of G, and $G^+(?Y)$ the *conclusion* [2]. A substitution for G is said to be *deciding* iff it binds only undecided variables of G (no free variables). When the conjunction $G^+(?Y)$ is empty, the implicative goal G is said to be *trivial.* When the conjunction G^- is empty, the goal G will be simply written as $G^+(?Y)$. Sometimes, we will use the special atom *true* to denote an empty conjunction of atoms. (By convention, a conjunction of atoms $B_1 \wedge ... \wedge B_m$ denotes the empty conjunction when $m = 0$.)

We assume given a definite logic program P defining the predicates which occur in goals. The variables that occur in the body but not in the head of a clause of P are called the *internal variables* of the clause. We will use the clauses of P in order to process implicative goals through the rules of the deduction system of *extended execution* [KS86] . The variables of the definite clauses in P are renamed at each use so that there occurs no conflict with the variable names of the implicative goals. The inference rules are applied in a backward manner: from a given implicative goal G the application of an inference rule generates a finite set of implicative goals $\{Hi\}_{i=1,...,n}$, such that G logically follows from the Hi's in the least Herbrand model of P. The process is iterated until all the goals newly generated are trivial.

An expression of the form $G\sigma(W, ?Y)$ denotes the instance of a formula G via a substitution σ, and indicates that $?Y$ is the vector of the undecided variables of $G\sigma$ and W a subvector of its free variables. Throughout the chapter, vectors of variables are denoted by the upper case letters U, V, W, X, Y, Z while individual variables are denoted by the lower case letters u, v, w, x, y, z,... Henceforth, the term "goal" will often be used as an abbreviation for "implicative goal".

Definition 1 *Let $G : G^+(?Y) \Leftarrow G^-$ be a goal, and X the vector of the free variables occurring in G. The* io-atom *associated with G is the atom $io_G(X, Y)$ where io_G denotes a new predicate symbol. The atom $io_G(X, Y)$ is abbreviated as $IO_G(Y)$, or sometimes more simply as IO_G.*

The position of the variables of X and Y as arguments of io_G is actually irrelevant, but fixed once given. In the examples of the chapter, the variables of a goal G will appear from-left-to-right as arguments of io_G according to their lexicographical ordering.

Example 1 [3] Let G be the goal: $append(u, v, ?y) \wedge length(?y, ?z)$. The free variables of G are u and v. The undecided variables are y and z. The associated *io*-atom is $io_G(u, v, y, z)$.

◇

Example 2 Let now G be the goal:

$$\times(?y, u, ?k) \wedge \times(?z, v, succ(?k)) \Leftarrow gcd(u, v, succ(0)).$$

The free variables of G are u and v, and the undecided variables k, y and z. The associated *io*-atom is $io_G(k, u, v, y, z)$.

◇

[2] The hypothesis (resp. conclusion) of G is denoted G^- (resp. G^+) because it is a "negative" (resp. "positive") subformula of G in the sense of, e.g., [MW80].

[3] The definitions of the predicates occurring in goals in the examples throughout the chapter are given in the appendix.

Definition 2 *Let $G : G^+(?Y) \Leftarrow G^-$ be a goal and Q a program defining the* io-*atom IO_G associated with G.*

1) *The program Q is* partially correct *for P w.r.t. G iff, in the least Herbrand model of $P \cup Q$,*
$$\forall Y\ (G^+(Y) \quad \Leftarrow \quad G^- \wedge IO_G(Y)).$$
2) *The program Q is* terminating [4] *for P w.r.t. G iff, in the least Herbrand model of $P \cup Q$,*
$$\exists Y\ IO_G(Y) \Leftarrow G^-.$$

Example 3 Let us consider the goal G: $append(u, v, ?y) \wedge length(?y, ?z)$. The statement of partial correctness w.r.t. G is, for all u and v:

$\forall y, z\ (append(u, v, y) \wedge length(y, z) \quad \Leftarrow \quad io_G(u, v, y, z))$.

The statement of termination is, for all u and v:

$\exists y, z\ io_G(u, v, y, z)$.

$\diamond$

Example 4 Let us now consider the goal

$G: \quad \times(?y, u, ?k) \wedge \times(?z, v, succ(?k)) \Leftarrow gcd(u, v, succ(0))$.

The statement of partial correctness w.r.t. G is, for all u and v:

$\forall k, y, z\ (\times(y, u, k) \wedge \times(z, v, succ(k)) \quad \Leftarrow \quad gcd(u, v, succ(0)) \wedge io_G(k, u, v, y, z))$.

The statement of termination is, for all u and v:

$\exists k, y, z\ io_G(u, v, y, z) \Leftarrow gcd(u, v, succ(0))$.

$\diamond$

Definition 3

1) *Given a one-step deduction δ of a goal G from a set of goals $\{Hi\}_{i=1,\dots,n}$, an* io-procedure associated with δ *is a logic program which defines IO_G in terms of $IO_{H1}, ..., IO_{Hn}$ (and also in terms of IO_G if the* io-*procedure is recursive). An* io-*procedure is made of definite clauses called* io-clauses.
2) *Given a deduction $\Delta : \delta_1, ..., \delta_m$ and* io-*procedures Q_{δ_j} associated with δ_j $(1 \leq j \leq m)$, the* io-program Q_Δ associated with Δ *is* $Q_{\delta_1} \cup ... \cup Q_{\delta_m}$.

Definition 4 *Let us consider a goal G and a deduction Δ of G from the set of goals $\{Hi\}_{i=1,\dots,n}$. Let Q_Δ be an* io-*program associated with this deduction. The program Q_Δ* preserves partial correctness *(resp.* preserves termination*) iff, for all program Q_{Hi} partially correct (resp. terminating) for P w.r.t Hi $(i = 1, ..., n)$, the program $Q_\Delta \cup Q_{H1} \cup ... \cup Q_{Hn}$ is partially correct (resp. terminating) for P w.r.t G.*

We are now going to explain how to synthesize a program Q that is partially correct and terminating for P w.r.t. a given implicative goal G by using the system of extended execution. Every inference rule of extended execution will be associated with a rule of construction for an *io*-procedure which preserves the properties of partial correctness and termination (see section 1). An *io*-program Q_Δ is thus associated with any deduction Δ of G from $\{Hi\}_{i=1,\dots,n}$. If we are given in addition programs Q_{Hi} that are partially correct and terminating w.r.t. Hi for all $1 \leq i \leq n$, then we are

[4] Q is terminating in the sense that a logic programming interpreter (with a complete search strategy) computes an answer in finite time to the query $\leftarrow IO_G\sigma(Y)$, for any ground substitution σ of the free variables of G such that $G^-\sigma$ holds.

done because $Q_\Delta \cup Q_{H1} \cup ... \cup Q_{Hn}$ is partially correct and terminating w.r.t. G and can thus be taken as Q. When the deduction Δ is a *proof*, i.e., when every goal Hi is trivial, the program made of the single unit clause IO_{Hi} will be taken as Q_{Hi} because it obviously satisfies the requirements of partial correctness and termination. (Actually, when a goal Hi is trivial, we will remove all the occurrences of IO_{Hi} from Q_Δ rather than add the unit definite clause IO_{Hi} to Q_Δ; this corresponds to adopting the convention that IO_{Hi} is the special atom $true$ when Hi is trivial.)

1.2

EXTENDED EXECUTION

In this section we review the main rules of extended execution [KS86] when applied to implicative goals, and define the associated *io*-procedures.

1.2.1

Definite Clause Inference (DCI)

The rule of Definite Clause Inference is a generalization of the rule of SLD-resolution (underlying Prolog) within the framework of extended execution: if a Prolog query $\leftarrow A(Y)$ has a successful SLD-derivation, then the corresponding implicative goal $A(?Y)$ has a proof by DCI in extended execution.

Definition 5 *Let G be a goal of the form $G^+(?Y) \Leftarrow G^-$. Let A be an atom in $G^+(?Y)$ and F a definite clause in P of the form $A' \leftarrow B_1, ..., B_m$. Suppose that A is unifiable with A' by a deciding mgu σ.*

1) Definite clause inference (DCI) *generates the singleton $\{H\}$, where H is the goal obtained from G by replacement of A with $B_1 \wedge ... \wedge B_m$, and instantiation via σ. All the new variables introduced into H are treated as undecided.*
2) *The* associated *io*-procedure *is*

$$\{ \ IO_G\sigma \leftarrow IO_H \ \}.$$

Example 5 Consider the goal

$H1: \quad append([], v, ?y) \wedge length(?y, ?z).$

DCI applies on $append([], v, ?y)$ via $\{y/v\}$ and yields $I1$: $length(v, ?z)$. The atom $io_{H1}(v, y, z)$ is defined by: $\quad io_{H1}(v, v, z) \leftarrow io_{I1}(v, z)$.

$\diamond$

Example 6 Consider the goal

$H2: \quad append([p|w], v, ?y) \wedge length(?y, ?z) \quad \Leftarrow \quad append(w, v, y') \wedge length(y', z').$

DCI applies on $append([p|w], v, ?y)$ via $\{y/[p|u]\}$ and yields

$I2: \quad append(w, v, ?u) \wedge length([p|?u], ?z)) \quad \Leftarrow \quad append(w, v, y') \wedge length(y', z').$

Then DCI applies again on $length([p|?u], ?z)$ via $\{z/succ(t)\}$ and yields

$J2: \quad append(w, v, ?u) \wedge length(?u, ?t) \quad \Leftarrow \quad append(w, v, y') \wedge length(y', z').$

The atom $io_{H2}(p, v, w, y, y', z, z')$ is defined by

$$io_{H2}(p, v, w, [p|u], y', z, z') \leftarrow io_{I2}(p, u, v, w, y', z, z').$$
$$io_{I2}(p, u, v, w, y', succ(t), z') \leftarrow io_{J2}(p, t, u, v, w, y', z').$$

$\diamond$

Proposition 6 *The* io-*procedure associated with the rule of DCI preserves partial correctness and termination.*

Proof
Partial correctness.
Let us first notice that the vectors of undecided variables in $G\sigma$ and H do not coincide in general. Let us denote by Z the vector of the undecided variables common to $G\sigma$ and H, and by U (resp. by V) the vector of the undecided variables proper to $G\sigma$ (resp. H). The existence of the variables of U explains as follows. Let $?Y^1$ be the subvector of the variables of $?Y$ that are proper to A in G and $?Y^2$ the complementary subvector. (The goal G thus writes as $A(?Y^1, ?Y^2) \wedge C(?Y^2) \Leftarrow G^-$.) Then $\sigma(Y^1)$ is a vector of terms, say $T^1(U, Z)$, in which the variables of U occur, while $\sigma(Y^2)$ is a vector of terms, say $T^2(Z)$, without any occurrence of a variable of U. The variables of the vector V are simply the internal variables of F. The instantiated clause $F\sigma$ is thus of the form $A(T^1(U, Z), T^2(Z)) \leftarrow F^-(V, Z)$, and the goal H is $F^-(?V, ?Z) \wedge C(T^2(?Z)) \Leftarrow G^-$. The associated *io*-procedure is $\{ \ IO_G(T^1(U, Z), T^2(Z)) \leftarrow IO_H(V, Z) \ \}$.

We have to show that if

$$F^-(V, Z) \wedge C(T^2(Z)) \quad \Leftarrow \quad G^- \wedge IO_H(V, Z) \qquad (Corr_H)$$

holds, then $A(Y^1, Y^2) \wedge C(Y^2) \quad \Leftarrow \quad G^- \wedge IO_G(Y^1, Y^2)$ holds. Let us suppose that $(Corr_H)$, G^- and $IO_G(Y^1, Y^2)$ hold, and let us show $A(Y^1, Y^2) \wedge C(Y^2)$. From $IO_G(Y^1, Y^2)$ and the implicit only-if part of the *io*-clause, it follows that Y^1 is equal to $T^1(U_0, Z_0)$ for some U_0 and Z_0, Y^2 is equal to $T^2(Z_0)$, and $IO_H(V_0, Z_0)$ holds for some V_0. From $IO_H(V_0, Z_0)$, G^- and $(Corr_H)$, it then follows $F^-(V_0, Z_0) \wedge C(T^2(Z_0))$. Using the instantiated clause $F\sigma$, we get $A(T^1(U, Z_0), T^2(Z_0)) \wedge C(T^2(Z_0))$, for all U. In particular, we have $A(T^1(U_0, Z_0), T^2(Z_0)) \wedge C(T^2(Z_0))$, that is $A(Y^1, Y^2) \wedge C(Y^2)$.

Termination.
We have to show that if

$$IO_H(V_0, Z_0) \Leftarrow G^- \qquad (Term_H)$$

holds for some V_0,Z_0, then $\exists Y^1, Y^2 \ IO_G(Y^1, Y^2) \Leftarrow G^-$ holds. Let us suppose that $(Term_H)$ and G^- hold, and let us show that $IO_G(Y_0^1, Y_0^2)$ holds for some Y_0^1 and Y_0^2. From $(Term_H)$ and G^- it follows $IO_H(V_0, Z_0)$. Then using the if-part of the *io*-clause, we get $IO_G(T^1(U, Z_0), T^2(Z_0))$ for any U. So we have $IO_G(Y_0^1, Y_0^2)$ by letting Y_0^1 be $T^1(U, Z_0)$ and Y_0^2 be $T^2(Z_0)$. ∎

1.2.2

Simplification

When applied to an implicative goal, the rule of simplification merely consists in unifying a conclusion atom against a hypothesis atom.

Definition 7 *Let G be a goal of the form $G^+(?Y) \Leftarrow G^-$. Let A be an atom of $G^+(?Y)$ and A' an atom of G^-. Suppose that A is unifiable to A' by a deciding mgu σ.*

1) Simplification *generates the singleton* $\{H\}$*, where* H *is obtained from* G *by elimination of* A *and (optionally)* A'*, and instantiation via* σ*.*
2) *The* associated *io*-procedure *is*
$$\{ \ IO_G\sigma \leftarrow IO_H \ \}.$$

Example 7 Consider the goal

$J2: \quad append(w, v, ?u) \wedge length(?u, ?t) \quad \Leftarrow \quad append(w, v, y') \wedge length(y', z').$

Simplification applies on $append(w, v, ?u)$ via $\{u/y'\}$ and yields

$K2: \quad length(y', ?t) \Leftarrow length(y', z').$

Again simplification applies on $length(y', ?t)$ via $\{t/z'\}$ and yields a trivial formula. The atom $io_{J2}(p, t, u, v, w, y', z')$ is defined by

$io_{J2}(p, t, y', v, w, y', z') \leftarrow io_{K2}(t, y', z').$

$io_{K2}(z', y', z').$

$\diamond$

Proposition 8 *The* io-*procedure associated with the rule of simplification preserves partial correctness and termination.*

The proof is similar to the proof of proposition 6 .

1.2.3

Restricted Structural Induction

So far the synthesis process associated with the first-order rules of DCI and simplification has merely mimicked the Prolog way of accumulating mgu's. Things are getting interesting with the introduction of an induction rule because the associated process then corresponds to the synthesis of a recursive function taking the free goal variables as arguments. The original system of extended execution [KF86, KS86] makes use of Clark's rule of computational induction [Cla79]. We postpone the study of this rule to the next subsection and first present the rule of structural induction [Bur69] which is easier to grasp. For the sake of conciseness, we present this rule only in case the induction variable is of type *list.* (The adaptation to other structures is easy.) The constructors of the list type are the empty list "[]" and the cons operator "[|]".

Definition 9 *Let* G *be the goal* $G^+(x, ?Y) \Leftarrow G^-$ *where* x *is a free variable of type* list *which occurs in* $G^+(?Y)$ *but not in* G^-*.*
1) (Restricted) structural induction *w.r.t.* x *generates the pair of goals*
$$H1: \quad G^+([], ?Y) \quad \Leftarrow \quad G^-.$$
$$H2: \quad G^+([p|w], ?Y) \quad \Leftarrow \quad G^+(w, Y') \wedge \ G^-.$$
where w *is a new free variable of type* list*,* p *is a new free variable of type* element *and* Y' *is a renaming of* Y *as a new vector of free variables.*
2) *The* associated *io*-atom $IO_G(x, Y)$ *is defined by:*
$$IO_G([], Y) \ \leftarrow \ IO_{H1}(Y).$$
$$IO_G([p|w], Y) \ \leftarrow \ IO_G(w, Y'), IO_{H2}(p, w, Y, Y').$$

Note that, in contrast with the classical induction scheme, the free variables of G distinct from x are not affected by the above induction rule and may be considered frozen as constants during the rule application. The rule thus corresponds to the *restricted* induction scheme of primitive recursive arithmetic [Sko67]. The definition of the associated *io*-atom IO_G is accordingly primitive recursive (see, e.g., [Rog67]). (Full classical induction would have required the free variables of G to be renamed and introduced as new undecided variables inside the hypothesis of $H2$, but occurrences of undecided variables within goal hypotheses are precluded in extended execution.)

Example 8 Consider G: $append(u, v, ?y) \wedge length(?y, ?z)$. By induction w.r.t. u, we have

$$H1: \quad append([], v, ?y) \wedge length(?y, ?z).$$
$$H2: \quad append([p|w], v, ?y) \wedge length(?y, ?z) \quad \Leftarrow \quad append(w, v, y') \wedge length(y', z').$$

The atom $io_G(u, v, y, z)$ is defined by

$$io_G([], v, y, z) \leftarrow io_{H1}(v, y, z).$$
$$io_G([p|w], v, y, z) \leftarrow io_G(w, v, y', z'), io_{H2}(p, v, w, y, y', z, z').$$

$\diamond$

Proposition 10 *The* io*-procedure associated with the rule of restricted structural induction preserves partial correctness and termination.*

Proof

Partial correctness.

We have to show that if

$$\forall Y \;\; G^+([], Y) \Leftarrow G^- \wedge IO_{H1}(Y) \qquad (Corr_1)$$

and

$$\forall Y, Y' \;\; G^+([p|w], Y) \;\; \Leftarrow \;\; G^+(w, Y') \wedge G^- \wedge IO_{H2}(p, w, Y, Y') \qquad (Corr_2)$$

hold, then

$$\forall Y \;\; G^+(x, Y) \Leftarrow G^- \wedge IO_G(x, Y) \qquad (Corr(x))$$

holds. Let us suppose $(Corr_1)$ and $(Corr_2)$ and let us show $(Corr(x))$ by structural induction w.r.t. x.

Base case ($x = []$). Let us show that G^- and $IO_G([], Y)$ imply $G^+([], Y)$. From $IO_G([], Y)$ and the only-if part of the first *io*-clause, we get $IO_{H1}(Y)$. From G^- and $(Corr_1)$ it then follows $G^+([], Y)$.

Induction step ($x = [p|w]$). Let us show that G^- and $IO_G([p|w], Y)$ imply $G^+([p|w], Y)$. From $IO_G([p|w], Y)$ and the only-if part of the second *io*-clause, we get $IO_{H2}(p, w, Y, Y'_0)$ and $IO_G(w, Y'_0)$ for some Y'_0. From G^- and the induction hypothesis $(Corr(w))$ it then follows $G^+(w, Y'_0)$. Then from $(Corr_2)$ it follows $G^+([p|w], Y)$.

Termination.

We have to show that if

$$\exists Y \;\; IO_{H1}(Y) \Leftarrow G^- \qquad (Term_1)$$

and

$$\forall Y' \, \exists Y \;\; IO_{H2}(p, w, Y, Y') \;\; \Leftarrow \;\; G^+(w, Y') \wedge G^- \qquad (Term_2)$$

hold, then

$$\exists Y \;\; IO_G(x, Y) \Leftarrow G^- \qquad (Term(x))$$

holds. Let us suppose $(Term_1)$ and $(Term_2)$ and let us show $(Term(x))$ by structural induction w.r.t. x.

Base case ($x = []$). Let us show that G^- implies $\exists Y \, IO_G([], Y)$. From G^- and $(Term_1)$ it follows $IO_{H1}(Y_0)$ for some Y_0. By the first *io*-clause, we get $IO_G([], Y_0)$.

Induction step ($x = [p|w]$). Let us show that G^- implies $\exists Y\, IO_G([p|w], Y)$. From G^- and the induction hypothesis ($Term(w)$) we get $IO_G(w, Y_0')$ for some Y_0'. By the partial correctness result, it then follows $G^+(w, Y_0')$. Then by ($Term_2$) we have $IO_{H2}(p, w, Y_0, Y_0')$ for some Y_0. Using the second *io*-clause, we finally obtain $IO_G([p|w], Y_0)$.

■

1.2.4

Computational Induction

In order to define the computational induction rule, it is convenient to first introduce the notion of "instantiated program" for an atom A [KF86]. Let $\{\ F_i\ \}_{i=1,\ldots,n}$ be the set of the clauses in P of the form $A_i' \leftarrow B_i$ such that the head A_i' unifies with A via the mgu $\sigma_i \cup \tau_i$ ($A_i'\sigma_i = A\tau_i$). The *instantiated (sub)program* of P for A is the set $\{$ $F_i\sigma_i\ \}_{i=1,\ldots,n}$. For the sake of simplicity, we assume throughout the chapter that the clauses in P are not mutually recursive. The body $B_i\sigma_i$ of any instantiated i-th clause can be then partitioned as a *recursive part* R_i and an *auxiliary part* S_i: the recursive part R_i is made of m_i atoms A_i^j ($1 \leq j \leq m_i$) which are instances of the atom A via a substitution θ_i^j ($A_i^j = A\theta_i^j$). The instantiated i-th clause $F_i\sigma_i$ can thus be alternatively written as $A\tau_i \ \leftarrow\ \bigwedge_{j=1}^{m_i} A\theta_i^j,\ S_i$.

Example 9 Consider the following definition of *gcd*

$F1:\quad gcd(a, a, a) \leftarrow a > 0.$
$F2:\quad gcd(a, b, d) \leftarrow gcd(c, b, d), +(c, b, a), a > b.$
$F3:\quad gcd(a, b, d) \leftarrow gcd(a, c, d), +(c, a, b), b > a.$

Let the atom A be $gcd(u, v, succ(0))$. Then we have:

$m_1 = 0, m_2 = m_3 = 1,$
$\tau_1 = \{u/succ(0), v/succ(0)\}, \sigma_1 = \{a/succ(0)\},$
$\tau_2 = \tau_3 = \{u/a, v/b\}, \sigma_2 = \sigma_3 = \{d/succ(0)\},$
$\theta_2 = \{u/c, v/b\}, \theta_3 = \{u/a, v/c\}$.

The instantiated program for $gcd(u, v, succ(0))$ is

$F1':\quad gcd(succ(0), succ(0), succ(0)) \leftarrow succ(0) > 0.$
$F2':\quad gcd(a, b, succ(0)) \leftarrow gcd(c, b, succ(0)), +(c, b, a), a > b.$
$F3':\quad gcd(a, b, succ(0)) \leftarrow gcd(a, c, succ(0)), +(c, a, b), b > a.$

◇

Definition 11 *Let G be a goal of the form $G^+(?Y) \Leftarrow A$, where A is an atom. Consider the instantiated program P' of P for A*

$$\{\ A\tau_i \ \leftarrow\ \bigwedge_{j=1}^{m_i} A\theta_i^j(V_i),\ S_i(V_i)\ \}_{i=1,\ldots,n}\ .$$

where V_i is the vector of the internal variables of the i-th clause of P'. Let $Y'^1, \ldots, Y'^{m_i}$ denote distinct renamings of Y as new vectors of free variables.

1) Computational induction (w.r.t. A) *generates the set $\{\ Hi\ \}_{i=1,\ldots,n}$, where Hi is defined by:*

$$G^+\tau_i(?Y) \ \Leftarrow\ \bigwedge_{j=1}^{m_i} G^+\theta_i^j(V_i, Y'^j) \wedge S_i(V_i)\ .$$

All the new variables introduced into the Hi's are treated as free.

2) The associated *io*-procedure *is:*
$\{IO_G\tau_i(Y) \leftarrow \bigwedge_{j=1}^{m_i} IO_G\theta_i^j(V_i, Y'^j), E_i(V_i), IO_{Hi}(V_i, Y, Y'^1, ..., Y'^{m_i})\}_{i=1,...,n}$,
where $E_i(V_i)$ *denotes, for all* $1 \leq i \leq n$, *an atom conjunction which*

- *i) contains only variables occurring in* $\bigwedge_{j=1}^{m_i} A\theta_i^j(V_i) \wedge S_i(V_i)$,
- *ii) contains at least one occurrence of every variable of* V_i, *and*
- *iii) satisfies in the least Herbrand model of* P *the equivalence subconditions:*

$$E_i(V_i) \Leftarrow \bigwedge_{j=1}^{m_i} A\theta_i^j(V_i) \wedge S_i(V_i). \qquad (e1_i)$$
$$\bigwedge_{j=1}^{m_i} A\theta_i^j(V_i) \wedge S_i(V_i) \Leftarrow A\tau_i \wedge E_i(V_i). \qquad (e2_i)$$

The computational induction scheme is thus obtained by replacing the instances of A in the instantiated program with the corresponding instances of G^+. The subexpressions $G^+\theta_i^j(V_i, Y'^j)$ of H_i play the same role as induction hypotheses in classical induction.

Note that the vector V_i of internal variables of the i-th clause of P' is also the vector of internal variables of the i-th clause of P since internal variables are not affected by σ_i. Note also that the free variables of the conclusion of G which do not occur in the hypothesis A are not affected by the above induction scheme and may be considered as frozen as constants during the rule application. (This variable freezing is the counterpart of freezing the free variables distinct from x in structural induction w.r.t. x, see section 1.)

There is a version of the computational induction rule (called "equivalence-preserving" computational induction, see [Fri88]) where the hypothesis of H_i contains in addition the atom conjunction $\bigwedge_{j=1}^{m_i} A\theta_i^j(V_i)$. The definition of the *io*-procedure for this version is identical to the former one.

The equivalence subconditions $(e1_i)$ and $(e2_i)$ express that E_i is an expression which makes the head and the body of the instantiated i-th clause equivalent $(A\tau_i \wedge E_i(V_i) \Leftrightarrow \bigwedge_{j=1}^{m_i} A\theta_i^j(V_i) \wedge S_i(V_i))$. The body of the i-th clause $\bigwedge_{j=1}^{m_i} A\theta_i^j(V_i) \wedge S_i(V_i)$ satisfies trivially the equivalence subconditions and can always be taken as $E_i(V_i)$. In practice there are often some proper subformulas of the body that satisfy $(e1_i)$ and $(e2_i)$ as well. (Note that any subformula of the body trivially satisfies $(e1_i)$.) A particular simple case happens when the heads of the clauses of the instantiated program are two-by-two nonunifiable and when there are no internal variables (V_i is empty): the trivial atom $true$ then satisfies the equivalence subconditions.

Example 10 Consider the following goal (which is nothing but a version of Bezout identity)

$G: \quad \times(?y, u, ?k) \wedge \times(?z, v, succ(?k)) \Leftarrow gcd(u, v, succ(0))$.

The instantiated program for $gcd(a, b, succ(0))$ is (see example 9)

$F1': \quad gcd(succ(0), succ(0), succ(0)) \leftarrow succ(0) > 0.$
$F2': \quad gcd(a, b, succ(0)) \leftarrow gcd(c, b, succ(0)), +(c, b, a), a > b.$
$F3': \quad gcd(a, b, succ(0)) \leftarrow gcd(a, c, succ(0)), +(c, a, b), b > a.$

Computational induction on G generates

$H1: \quad \times(?y, succ(0), ?k) \wedge \times(?z, succ(0), succ(?k)) \Leftarrow succ(0) > 0.$
$H2: \quad \times(?y, a, ?k) \wedge \times(?z, b, succ(?k))$
$\qquad\qquad \Leftarrow \times(y', c, k') \wedge \times(z', b, succ(k')) \wedge +(c, b, a) \wedge a > b.$
$H3: \quad \times(?y, a, ?k) \wedge \times(?z, b, succ(?k))$
$\qquad\qquad \Leftarrow \times(y', a, k') \wedge \times(z', c, succ(k')) \wedge +(c, a, b) \wedge b > a.$

The expressions $true$, $+(c,b,a) \wedge a > b$ and $+(c,a,b) \wedge b > a$ are subformulas of the body of $F1'$, $F2'$ and $F3'$ respectively and thus satisfy their subconditions (*e1*). On the other hand, they satisfy their respective subconditions (*e2*) since

$succ(0) > 0 \quad \Leftarrow \quad gcd(succ(0), succ(0), succ(0)) \wedge true.$

$gcd(c,b,succ(0)) \wedge +(c,b,a) \wedge a > b \quad \Leftarrow \quad gcd(a,b,succ(0)) \wedge +(c,b,a) \wedge a > b.$

$gcd(a,c,succ(0)) \wedge +(c,a,b) \wedge b > a \quad \Leftarrow \quad gcd(a,b,succ(0)) \wedge +(c,a,b) \wedge b > a.$

The associated *io*-procedure $io_G(a,b,k,y,z)$ is thus:

$io_G(succ(0), succ(0), k, y, z) \quad \leftarrow \quad io_{H1}(k,y,z).$

$io_G(a,b,k,y,z) \leftarrow io_G(c,b,k',y',z'), io_{H2}(a,b,c,k,k',y,y',z,z'), +(c,b,a), a > b.$

$io_G(a,b,k,y,z) \leftarrow io_G(a,c,k',y',z'), io_{H3}(a,b,c,k,k',y,y',z,z'), +(c,a,b), b > a.$

◇

Proposition 12 *The* io*-procedure associated with the rule of computational induction preserves partial correctness and termination.*

Proof

Partial correctness.

For the sake of notation simplicity we will assume that every clause of the instantiated program contains one and only one occurrence of A in its body (i.e., $m_i = 1$ for all $1 \leq i \leq n$). We have to show that if, for all $1 \leq i \leq n$,

$$G^+\tau_i(Y) \;\Leftarrow\; G^+\theta_i(V_i, Y') \wedge S_i(V_i) \wedge IO_{Hi}(V_i, Y, Y') \qquad (Corr_i)$$

holds, then

$$(G^+(Y) \Leftarrow A) \Leftarrow IO_G(Y) \qquad (Corr_G)$$

holds. Let us suppose $(Corr_i)$ for all $1 \leq i \leq n$, and let us show $(Corr_G)$ by computational induction w.r.t. IO_G, that is let us prove

$$G^+\tau_i(Y) \Leftarrow A\tau_i$$

assuming $E_i(V_i)$, $IO_{Hi}(V_i, Y, Y')$ and the "induction hypothesis":

$$G^+\theta_i(V_i, Y') \Leftarrow A\theta_i(V_i) \;.$$

If $A\tau_i$ is true, then, using $E_i(V_i)$ and the equivalence subcondition $(e2_i)$, we have $S_i(V_i)$ and $A\theta_i(V_i)$. Then from the induction hypothesis we get $G^+\theta_i(V_i, Y')$. From $S_i(V_i)$, $IO_{Hi}(V_i, Y, Y')$ and $(Corr_i)$ it then follows $G^+\tau_i(Y)$.

Termination.

We have to show that if, for all $1 \leq i \leq n$,

$$\exists Y \; IO_{Hi}(V_i, Y, Y') \;\Leftarrow\; G^+\theta_i(V_i, Y') \wedge S_i(V_i) \qquad (Term_i)$$

holds, then

$$\exists Y \quad IO_G(Y) \Leftarrow A \qquad (Term)$$

holds. Let us suppose $(Term_i)$ for all $1 \leq i \leq n$, and let us show $(Term)$ by computational induction w.r.t. A, that is let us prove

$$\exists Y \, IO_G\tau_i(Y)$$

assuming $S_i(V_i)$, $A\theta_i(V_i)$ and the "induction hypothesis":

$$IO_G\theta_i(V_i, Y_0') \quad \text{for some } Y_0'.$$

From $IO_G\theta_i(V_i, Y_0')$, $A\theta_i(V_i)$ and the partial correctness result, it follows $G^+\theta_i(V_i, Y_0')$. Then using $S_i(V_i)$ and $(Term_i)$ we get $IO_{Hi}(V_i, Y_0, Y_0')$ for some Y_0. On the other hand from $S_i(V_i), A\theta_i(V_i)$ and the equivalence subcondition $(e1_i)$ we have $E_i(V_i)$. From $E_i(V_i)$, $IO_{Hi}(V_i, Y_0, Y_0')$, $IO_G\theta_i(V_i, Y_0')$ and the i-th *io*-clause we get $IO_G\tau_i(Y_0)$.

■

1.2.5

Negation as Failure Inference (NFI)

The negation as failure inference rule (NFI) proceeds by replacing an atom A of the goal hypothesis which unifies with a P-clause head by the corresponding instance of the P-clause body. There are as many derived goals as possible unifications. In case A unifies with no P-clause head, the set of derived goals is empty and is thus vacuously true. The rule can be seen as an adaptation of Clark's rule of negation as failure used in Prolog [Cla78, KS86] into the framework of extended execution: if a ground Prolog query $\leftarrow \neg A$ has a successful derivation via Clark's negation as failure, then the corresponding implicative goal $false \Leftarrow A$ has a proof via NFI in extended execution [Kan88].

Definition 13 *Let G be a goal of the form $G^+(?Y) \Leftarrow A \wedge C$, where A is an atom and C a conjunction of atoms. Consider the instantiated program P' of P for A*

$$\{ \; A\tau_i \quad \leftarrow \quad B_i\sigma_i(V_i) \; \}_{i=1,\ldots,n}$$

where V_i is the vector of the internal variables of the i-th clause of P'.

1) Negation as failure inference (NFI) *generates the set $\{ \; Hi \; \}_{i=1,\ldots,n}$, where Hi is defined by:*

$$G^+\tau_i(?Y) \quad \Leftarrow \quad B_i\sigma_i(V_i) \wedge C\tau_i \; .$$

All the new variables introduced into the Hi's are treated as free.

2) The associated io-procedure *is:*

$$\{ \; IO_G\tau_i(Y) \quad \leftarrow \quad E_i(V_i), IO_{Hi}(V_i, Y) \; \}_{i=1,\ldots,n},$$

where $E_i(V_i)$ denotes, for all $1 \leq i \leq n$, an atom conjunction which

i) contains only variables occurring in $B_i\sigma_i(V_i)$,

ii) contains at least one occurrence of every variable of V_i,

iii) satisfies in the least Herbrand model of P the equivalence subconditions:

$$E_i(V_i) \quad \Leftarrow \quad B_i\sigma_i(V_i). \qquad (e1_i)$$

$$B_i\sigma_i(V_i) \quad \Leftarrow \quad A\tau_i \wedge E_i(V_i). \qquad (e2_i)$$

In the above definition the replaced atom A appears at the leftmost position in the hypothesis of G only to fix the ideas, but could actually occur anywhere in the hypothesis. In the case where the conjunction C in the body of G is empty, the NFI rule can be seen as a degenerate case of computational induction where the recursive parts of the P'-clauses are empty.

As in computational induction, the equivalence subconditions $(e1_i)$ and $(e2_i)$ express that E_i is an expression which makes the head and the body of the instantiated i-th clause equivalent $(A\tau_i \wedge E_i(V_i) \quad \Leftrightarrow \quad B_i\sigma_i(V_i))$. Here again the body of the i-th clause $B_i\sigma_i(V_i)$ trivially satisfies these subconditions and can always be taken as $E_i(V_i)$. Also in practice there are often some proper subformulas of $B_i\sigma_i(V_i)$ that satisfy $(e1_i)$ and $(e2_i)$ as well.

Example 11 Consider the goal

$G: \quad \times(?y, u, ?k) \wedge \times(?z, v, succ(?k)) \Leftarrow gcd(u, v, succ(0)).$

The instantiated program for $gcd(u, v, succ(0))$ (see example 9) is

$F1': \quad gcd(succ(0), succ(0), succ(0)) \leftarrow succ(0) > 0.$

$F2': \quad gcd(a, b, succ(0)) \leftarrow gcd(c, b, succ(0)), +(c, b, a), a > b.$

$F3'$: $gcd(a, b, succ(0)) \leftarrow gcd(a, c, succ(0)), +(c, a, b), b > a.$

NFI on G generates

$H1$: $\times(?y, succ(0), ?k) \wedge \times(?z, succ(0), succ(?k)) \Leftarrow succ(0) > 0.$

$H2$: $\times(?y, a, ?k) \wedge \times(?z, b, succ(?k)) \Leftarrow gcd(c, b, succ(0)) \wedge +(c, b, a) \wedge a > b.$

$H3$: $\times(?y, a, ?k) \wedge \times(?z, b, succ(?k)) \Leftarrow gcd(a, c, succ(0)) \wedge +(c, a, b) \wedge b > a.$

The expressions $true$, $+(c, b, a) \wedge a > b$ and $+(c, a, b) \wedge b > a$ satisfy the equivalence subconditions w.r.t. $F1'$, $F2'$ and $F3'$ respectively (see example 10). The associated *io*-procedure $io_G(a, b, k, y, z)$ is thus:

$io_G(succ(0), succ(0), k, y, z) \leftarrow io_{H1}(k, y, z).$

$io_G(a, b, k, y, z) \leftarrow io_{H2}(a, b, c, k, y, z), +(c, b, a), a > b.$

$io_G(a, b, k, y, z) \leftarrow io_{H3}(a, b, c, k, y, z), +(c, a, b), b > a.$

$\diamond$

Proposition 14 *The* io-*procedure associated with the rule of NFI preserves partial correctness and termination.*

The proof can be easily adapted from that of proposition 12 .

1.2.6

Hypothesis Replacement

The rule of hypothesis replacement that we are going to present now is not explicitly stated as a rule of the system of extended execution defined in [KF86, KS86]. Still such a rule is used in [KF86] (section 5) in order to replace a multiatomic goal hypothesis with an equivalent single atom *newp* and perform computational induction w.r.t. the recursive definition of *newp*. The rule of hypothesis replacement does not make use of definite clause(s) of P as in the DCI, NFI and computational induction rules, but makes use instead of an implicative goal L which holds in the least Herbrand model of P. Such a goal L is called a *lemma*. The variables of L are renamed at each use so that there occurs no conflict with goal variable names. We will distinguish the case where L is a "free" lemma, that is a lemma without undecided variables, and the case where L is a "general" lemma, that is a lemma that may contain undecided variables.

1.2.6.1

HYPOTHESIS REPLACEMENT VIA A FREE LEMMA

Definition 15 *Let L be the free lemma $B \Leftarrow D'$, and G the goal $G^+(?Y) \Leftarrow C \wedge D$, where C and D are conjunctions of atoms. Suppose that D matches with D' via the substitution σ ($D = D'\sigma$).*

1) Hypothesis replacement via the free lemma L *generates the singleton* $\{H\}$*, where* H *is*

$$G^+(?Y) \Leftarrow C \wedge B\sigma \ .$$

All the new variables introduced into H are treated as free.

2) *The* associated *io*-procedure *is*

$$\{ \ IO_G(Y) \leftarrow IO_H(Y) \ \}.$$

In the above definition the atoms of the replaced conjunction D appear at the rightmost position in the hypothesis of G only to fix the ideas, but could actually occur anywhere in the hypothesis. The (free) variables of L which occur in B but not in D' are not affected by σ and are the (free) variables which are introduced from G to H by the rule application.

The rule of hypothesis deletion [Fri88] which removes some atoms of the hypothesis of a goal is a particular case of the above rule of hypothesis replacement (B is the atom $true$). The already mentioned process used in [KF86], which replaces a hypothesis with an equivalent atom *newp*, can also be seen as an application of the above rule.

Proposition 16 *The* io-*procedure associated with the rule of hypothesis replacement via a free lemma preserves partial correctness and termination.*

The proof is easy and is omitted.

1.2.6.2

HYPOTHESIS REPLACEMENT VIA A GENERAL LEMMA

Definition 17 *Let L be the general lemma $B(?V) \Leftarrow D'$, and G the goal $G^+(?Y) \Leftarrow C \wedge D$, where C and D are conjunctions of atoms. Suppose that D matches with D' via the substitution σ ($D = D'\sigma$).*

1) Hypothesis replacement via the general lemma L *generates the singleton* $\{H\}$, *where H is*

$$G^+(?Y) \quad \Leftarrow \quad C \wedge B\sigma(V) \ .$$

All the new variables introduced into H are treated as free.

2) *The* associated *io*-procedure *is*

$$\{ \ IO_G(Y) \ \leftarrow \ E(V), IO_H(V,Y) \ \},$$

where $E(V)$ denotes an atom conjunction which

i) *contains only variables occurring in $B\sigma(V)$,*

ii) *contains at least one occurrence of every variable of V,*

iii) *satisfies in the least Herbrand model of P the equivalence subconditions:*

$$E(V) \quad \Leftarrow \quad B\sigma(V). \qquad (e1)$$

$$B\sigma(V) \quad \Leftarrow \quad D \wedge E(V). \qquad (e2)$$

As in the previous definition the atoms of the conjunction D appear at the rightmost position in the hypothesis of G only to fix the ideas, but could actually occur anywhere in the hypothesis. Note that the undecided variables of $?V$ in L are not affected by σ, and that they occur as free variables in H. The free variables of L which occur in $B(?V)$ but not in D' are neither affected by σ. These variables together with those of V are the (free) variables introduced from G to H by the rule application.

In contrast with what happened with computational induction and NFI rules, the expression $E(V)$ does not make equivalent the "head" and the "body" of the instantiated lemma $L\sigma$. Here it is the body D which makes the head $B\sigma(V)$ and the expression $E(V)$ equivalent ($D \wedge B\sigma(V) \Leftrightarrow D \wedge E(V)$). The atom conjunction $B\sigma(V)$ satisfies trivially the equivalence subconditions ($e1$) and ($e2$) and can always be taken as $E(V)$. In practice, there are often some proper subformulas of $B\sigma(V)$ that satisfy ($e1$) and ($e2$) as well. (Here again any subformula of $B\sigma(V)$ trivially satisfies (*e1*).)

Example 12 Consider the goals

$$H2: \quad \times(?y,a,?k) \wedge \times(?z,b,succ(?k)) \Leftarrow \times(y',c,k') \wedge \times(z',b,succ(k')) \wedge +(c,b,a) \wedge a > b.$$

$$H3: \quad \times(?y,a,?k) \wedge \times(?z,b,succ(?k)) \Leftarrow \times(y',a,k') \wedge \times(z',c,succ(k')) \wedge +(c,a,b) \wedge b > a.$$

Consider the lemmas

$$L2: \quad \exists k'',z'' \ \times(q,f,k'') \wedge \times(z'',g,succ(k'')) \wedge +(q,s,z'') \Leftarrow \times(q,h,r) \wedge \times(s,g,succ(r)) \wedge +(h,g,f).$$

$$L3: \quad \exists k'',z'' \ \times(y'',f,k'') \wedge \times(s,g,succ(k'')) \wedge +(q,s,y'') \Leftarrow \times(q,f,r) \wedge \times(s,h,succ(r)) \wedge +(h,f,g).$$

(More readably these lemmas could be expressed under a functional notation as $1+q\times(h+g) = (q+s)\times g \Leftarrow 1+q\times h = s\times g$ and as $1+(q+s)\times f = s\times(h+f) \Leftarrow 1+q\times f = s\times h$.) In $H2$ the hypothesis subpart $\times(y',c,k')\wedge\times(z',b,succ(k'))\wedge+(c,b,a)$ is an instance of the $L2$ hypothesis via the substitution $\{f/a, g/b, h/c, q/y', r/k', s/z'\}$. Likewise in $H3$ the hypothesis subpart $\times(y',a,k') \wedge \times(z',c,succ(k')) \wedge +(c,a,b)$ is an instance of the $L3$ hypothesis via the same substitution. The corresponding instantiated forms of $L2$ and $L3$ are respectively

$$L2': \quad \exists k'',z'' \ \times(y',a,k'') \wedge \times(z'',b,succ(k'')) \wedge +(y',z',z'') \Leftarrow \times(y',c,k') \wedge \times(z',b,succ(k')) \wedge +(c,b,a).$$

$$L3': \quad \exists k'',z'' \ \times(y'',a,k'') \wedge \times(z',b,succ(k'')) \wedge +(y',z',y'') \Leftarrow \times(y',a,k') \wedge \times(z',c,succ(k')) \wedge +(c,a,b).$$

By hypothesis replacement one thus transforms $H2$ and $H3$ into

$$I2: \quad \times(?y,a,?k) \wedge \times(?z,b,succ(?k)) \Leftarrow \times(y',a,k'') \wedge \times(z'',b,succ(k'')) \wedge +(y',z',z'') \wedge a > b.$$

$$I3: \quad \times(?y,a,?k) \wedge \times(?z,b,succ(?k)) \Leftarrow \times(y'',a,k'') \wedge \times(z',b,succ(k'')) \wedge +(y',z',y'') \wedge b > a.$$

The conjunctions $\times(y',a,k'')\wedge+(y',z',z'')$ and $\times(y'',a,k'')\wedge+(y',z',y'')$ are subparts of the conclusions of $L2'$ and $L3'$ respectively, thus satisfy their (*e1*) subconditions. On the other hand they satisfy their respective (*e2*) subconditions since

$$\times(y',a,k'') \wedge \times(z'',b,succ(k'')) \wedge +(y',z',z'') \Leftarrow \times(y',c,k') \wedge \times(z',b,succ(k')) \wedge +(c,b,a) \wedge \times(y',a,k'') \wedge +(y',z',z'').$$

and

$$\times(y'',a,k'') \wedge \times(z',b,succ(k'')) \wedge +(y',z',y'') \Leftarrow \times(y',a,k') \wedge \times(z',c,succ(k')) \wedge +(c,a,b) \wedge \times(y'',a,k'') \wedge +(y',z',y'').$$

We get the associated *io*-procedures

$$io_{H2}(a,b,c,k,k',y,y',z,z') \leftarrow io_{I2}(a,b,k,k'',y,y',z,z',z''), \times(y',a,k''), +(y',z',z'').$$

$$io_{H3}(a,b,c,k,k',y,y',z,z') \leftarrow io_{I3}(a,b,k,k'',y,y',y'',z,z'), \times(y'',a,k''), +(y',z',y'').$$

$\diamond$

Proposition 18 *The* io*-procedure associated with the rule of hypothesis replacement via a general lemma preserves partial correctness and termination.*

Proof

Partial correctness.

We have to show that $G^+(Y)$ holds assuming C, D, $IO_G(Y)$ and

$$G^+(Y) \;\Leftarrow\; C \wedge B\sigma(V) \wedge IO_H(V,Y). \qquad (Corr_H)$$

From $IO_G(Y)$ and the only-if part of the *io*-clause, it follows that $E(V_0)$ and $IO_H(V_0,Y)$ hold for some V_0. From D, $E(V_0)$ and $(e2)$ we get $B\sigma(V_0)$. Then using C, $IO_H(V_0,Y)$ and $(Corr_H)$ we have $G^+(Y)$.

Termination.
We have to show that $\exists Y\ IO_G(Y)$ holds assuming C, D and, for all V,

$$IO_H(V,Y_V) \;\Leftarrow\; C \wedge B\sigma(V) \quad \text{for some } Y_V. \qquad (Term_H(V))$$

From D, using L and the fact that $D = D'\sigma$, it follows $B\sigma(V_0)$ for some V_0. By $B\sigma(V_0)$, C and $(Term_H(V_0))$ we have $IO_H(V_0, Y_{V_0})$. On the other hand, by $B\sigma(V_0)$ and $(e1)$ we have $E(V_0)$. From $E(V_0)$, $IO_H(V_0, Y_{V_0})$ and the if-part of the *io*-clause, we then get $IO_G(Y_{V_0})$. ■

1.2.7

Postulation

This rule is not a rule of extended execution. It aims at stopping the development of a deduction branch by "postulating" the truth of the current goal (without giving a proof). Formally we have

Definition 19 *Let G be a goal of the form $C(?Y)$ where $C(?Y)$ is a conjunction of atoms.*

1) Postulation *generates the trivial goal.*
2) *The* associated *io*-procedure *is*

$$\{\ IO_G(Y) \leftarrow C(Y)\ \}.$$

Example 13 Consider the goal $I1 : length(v, ?z)$ (see example 5). This goal is a lemma (which intuitively says that the "function" *length* is defined everywhere). Postulation generates the trivial goal and the associated *io*-procedure is

$$io_{I1}(v,z) \leftarrow length(v,z)\ .$$

◇

Proposition 20 *The* io*-procedure associated with the rule of postulation applied to a goal G preserves partial correctness and termination provided that G is a lemma.*

The proof is easy and is omitted.

1.3

CORRECTNESS

From the propositions of the former section on the preservation of partial correctness and termination and from the partial correctness and termination properties of the *io*-procedures associated with trivial goals, it follows immediately

Theorem 21 *Let G be a goal and Δ a deduction of G from trivial goals using the rules of DCI, simplification, NFI, restricted structural induction, computational induction, hypothesis replacement and postulation. The* io-program *associated with Δ is partially correct and terminating w.r.t. G provided that postulation is applied to lemmas only.*

Example 14 The examples 5, 6, 7, 8 and 13 show that there is a deduction Δ of G: $append(u, v, ?y) \wedge length(?y, ?z)$ from trivial goals. The *io*-atom IO_G is defined by the following program

$io_G([], v, y, z) \leftarrow io_{H1}(v, y, z).$
$io_{H1}(v, v, z) \leftarrow io_{I1}(v, z).$
$io_{I1}(v, z) \leftarrow length(v, z).$
$io_G([p|w], v, y, z) \leftarrow io_G(w, v, y', z'), io_{H2}(p, v, w, y, y', z, z').$
$io_{H2}(p, v, w, [p|u], y', z, z') \leftarrow io_{I2}(p, u, v, w, y', z, z').$
$io_{I2}(p, u, v, w, y', succ(t), z') \leftarrow io_{J2}(p, t, u, v, w, y', z').$
$io_{J2}(p, t, y', v, w, y', z') \leftarrow io_{K2}(t, y', z').$
$io_{K2}(z', y', z').$

By unfolding [TS84], this program is transformed under the equivalent form Q:

$io_G([], v, v, z) \leftarrow length(v, z).$
$io_G([p|u], v, [p|y], succ(z)) \leftarrow io_G(u, v, y, z).$

The program Q is partially correct w.r.t. G, i.e., for all u and v:

$$\forall y, z\ append(u, v, y) \wedge length(y, z) \quad \Leftarrow \quad io_G(u, v, y, z).$$

The program Q is terminating w.r.t. G, i.e., for all u and v:

$$\exists y, z\ io_G(u, v, y, z).$$

Note that the program Q defining $io_G(u, v, y, z)$ is identical to the program defining $newp(u, v, y, z)$ that is obtained by applying the unfolding/folding transformational method [TS84] to the clause $newp(u, v, y, z) \leftarrow append(u, v, y), length(y, z)$.

$\diamond$

Example 15 Let us give a constructive proof of a version of Bezout identity expressed as

$$G: \quad \times(?y, u, ?k) \wedge \times(?z, v, succ(?k)) \Leftarrow gcd(u, v, succ(0)).$$

By computational induction (see example 10) one generates

$$H1: \quad \times(?y, succ(0), ?k) \wedge \times(?z, succ(0), succ(?k)) \quad \Leftarrow \quad succ(0) > 0.$$
$$H2: \quad \times(?y, a, ?k) \wedge \times(?z, b, succ(?k)) \\ \Leftarrow \quad \times(y', c, k') \wedge \times(z', b, succ(k')) \wedge +(c, b, a) \wedge a > b.$$
$$H3: \quad \times(?y, a, ?k) \wedge \times(?z, b, succ(?k)) \\ \Leftarrow \times(y', a, k') \wedge \times(z', c, succ(k')) \wedge +(c, a, b) \wedge b > a.$$

The associated *io*-procedure $io_G(u, v, k, y, z)$ is:

$io_G(succ(0), succ(0), k, y, z) \quad \leftarrow \quad io_{H1}(k, y, z).$
$io_G(a, b, k, y, z) \quad \leftarrow$
$\qquad io_G(c, b, k', y', z'), io_{H2}(a, b, c, k, k', y, y', z, z'), +(c, b, a), a > b.$
$io_G(a, b, k, y, z) \quad \leftarrow$
$\qquad io_G(a, c, k', y', z'), io_{H3}(a, b, c, k, k', y, y', z, z'), +(c, a, b), b > a.$

Using the definitions of $\times$ and $+$, we can transform $H1$ into a trivial goal by applying two DCIs via the substitutions $\{y/0, k/0\}$ and $\{z/succ(0)\}$. The associated *io*-procedure is (after unfolding)

$io_{H1}(0, 0, succ(0)).$

By hypothesis replacement (see example 12) one transforms respectively $H2$ and $H3$

into

$$I2: \quad \times(?y,a,?k) \wedge \times(?z,b,succ(?k)) \Leftarrow \times(y',a,k'') \wedge \times(z'',b,succ(k'')) \wedge +(y',z',z'') \wedge a>b.$$

$$I3: \quad \times(?y,a,?k) \wedge \times(?z,b,succ(?k)) \Leftarrow \times(y'',a,k'') \wedge \times(z',b,succ(k'')) \wedge +(y',z',y'') \wedge b>a.$$

The associated *io*-procedures are

$$io_{H2}(a,b,c,k,k',y,y',z,z') \leftarrow io_{I2}(a,b,k,k'',y,y',z,z',z''), \times(y',a,k''), +(y',z',z'').$$

$$io_{H3}(a,b,c,k,k',y,y',z,z') \leftarrow io_{I3}(a,b,k,k'',y,y',y'',z,z'), \times(y'',a,k''), +(y',z',y'').$$

By simplification via the substitution $\{y/y', z/z'', k/k''\}$ one can transform $I2$ into a trivial formula. Likewise, by simplification via the substitution $\{y/y'', z/z', k/k''\}$ one can transform $I3$ into a trivial formula. The associated *io*-procedures are

$$io_{I2}(a,b,k'',k'',y',y',z'',z',z'').$$

$$io_{I3}(a,b,k'',k'',y'',y',y'',z',z').$$

Finally, after unfolding, we get as *io*-program

$$io_G(succ(0), succ(0), 0, 0, succ(0)) \quad .$$

$$io_G(a,b,k,y,z) \;\leftarrow\; io_G(c,b,k',y,z'), +(c,b,a), a>b, +(y,z',z), \times(y,a,k).$$

$$io_G(a,b,k,y,z) \;\leftarrow\; io_G(a,c,k',y',z), +(c,a,b), b>a, +(y',z,y), \times(y,a,k).$$

The property of partial correctness w.r.t. G states

$$\forall u,v,k,y,z \;\; (\times(y,u,k) \wedge \times(z,v,succ(k)) \;\Leftarrow\; gcd(u,v,succ(0)) \wedge io_G(k,u,v,y,z)).$$

The property of termination states

$$\forall u,v \; \exists k,y,z \;\; io_G(u,v,y,z) \Leftarrow gcd(u,v,succ(0)).$$

From these two properties it follows that, given two numbers u_0 and v_0 satisfying $gcd(u_0,v_0,succ(0)))$, a complete logic interpreter submitted with the query $\leftarrow io_G(u_0,v_0,k,y,z)$ gives in finite time answers k_0, y_0 and z_0 such that

$$\times(y_0,u_0,k_0) \wedge \times(z_0,v_0,succ(k_0)).$$

In other words, given two pairwise prime numbers u_0 and v_0 as inputs, the extracted program computes two numbers y_0 and z_0 satisfying the Bezout identity $z_0 \times v_0 - y_0 \times u_0 = 1$.

◇

1.4

IO-ATOM PROJECTION

The *io*-procedures keep track of all the variables occurring in every goal of the deduction, and thus often convey more information than is necessary, which is a source of inefficiency. A simple remedy to this defect consists in "projecting" [5] *io*-atoms onto the useful arguments and dropping the superfluous arguments.

Definition 22

[5] In the Conference version of this chapter (Proc. 7th Intl. Conf. on Logic Programming, Jerusalem, 1990) we used the term "truncation" instead of "projection", but the latter seems to be more standard terminology (see, e.g., [PP89]).

1) *Let G be a goal, $V_1 \cup V_2$ the vector of its variables and $io_G(V_1, V_2)$ its associated* io-*atom. Given a new predicate symbol io'_G, we say that $io'_G(V_2)$ is the* projection *of $io_G(V_1, V_2)$ onto V_2. By convention, io'_G is the special atom* true *when V_2 is empty.*
2) *Let Q be a program defining the* io-*atom $io_G(V_1, V_2)$. We say that the program Q' is a* projected form *of Q onto V_2 if Q' is obtained from Q by replacing all the instances of $io_G(V_1, V_2)$ by the corresponding instances of $io'_G(V_2)$ and all the instances of every auxiliary atom, say $io_H(W_1, W_2)$, by the corresponding instances of a projection $io'_H(W_2)$.*

The position of the elements of the projection vector V_2 (resp. W_2) as arguments of io'_G (resp. io'_H) is actually irrelevant. (They do not need to occur at rightmost positions.) For any auxiliary predicate io_H, the choice of the projection vector W_2 is free: one can either let the atom $io_H(W_1, W_2)$ as it is (W_1 is empty), or replace it by the atom $true$ (W_2 is empty), or take any intermediate choice (W_1 and W_2 are non-empty). Projection applies in turn to the auxiliary atoms involved in the recursive definition of io_H. Strategies of projection are discussed in [Bou91a].

Example 16 Consider the program Q of example 14 defining $io_G(u, v, y, z)$:

$$io_G([], v, v, z) \leftarrow length(v, z).$$
$$io_G([p|u], v, [p|y], succ(z)) \leftarrow io_G(u, v, y, z).$$

A projected form of Q onto the first, second and fourth arguments of io_G is Q':

$$io'_G([], v, z) \leftarrow length(v, z).$$
$$io'_G([p|u], v, succ(z)) \leftarrow io'_G(u, v, z).$$

Note that the program Q' defining $io'_G(u, v, z)$ is identical to the program defining $newp'(u, v, z)$ that is obtained by applying the unfolding/folding transformational method to

$$newp'(u, v, z) \quad \leftarrow \quad append(u, v, y), length(y, z).$$

◇

Example 17 Consider the program of example 15 defining $io_G(u, v, k, y, z)$:

$$io_G(succ(0), succ(0), 0, 0, succ(0)) \quad .$$
$$io_G(a, b, k, y, z) \quad \leftarrow \quad io_G(c, b, k', y, z'), +(c, b, a), a > b, +(y, z', z), \times(y, a, k).$$
$$io_G(a, b, k, y, z) \quad \leftarrow \quad io_G(a, c, k', y', z), +(c, a, b), b > a, +(y', z, y), \times(y, a, k).$$

A projected form of the program onto the first, second, fourth and fifth arguments of io_G is

$$io'_G(succ(0), succ(0), 0, succ(0)) \quad .$$
$$io'_G(a, b, y, z) \quad \leftarrow \quad io'_G(c, b, y, z'), +(c, b, a), a > b, +(y, z', z).$$
$$io'_G(a, b, y, z) \quad \leftarrow \quad io'_G(a, c, y', z), +(c, a, b), b > a, +(y', z, y).$$

This program defines $io'_G(u, v, y, z)$. Note that the auxiliary atom $\times(y, a, k)$ has been deleted (replaced by $true$) in the second and third clauses.

◇

We now define the notions of partial correctness and termination for projected forms of *io*-programs. We assume that G is a goal of the form $G^+(?Y, ?Z) \Leftarrow G^-$ and Q is the program defining the associated *io*-atom $IO_G(Y, Z)$. We consider a projected form Q' of Q onto Z which defines the projection, say $IO'_G(Z)$, of $IO_G(Y, Z)$. (The expression $IO'_G(Z)$ abbreviates for $io'_G(X, Z)$ where X denotes the vector of the free variables of G.)

Definition 23

1) *The program Q' is* partially correct w.r.t. G *iff, in the least Herbrand model of $P \cup Q'$,*
$$\forall Z\ \exists Y\ G^+(Y, Z) \quad \Leftarrow \quad G^- \wedge IO'_G(Z).$$
2) *The program Q' is* terminating w.r.t. G *iff, in the least Herbrand model of $P \cup Q'$,*
$$\exists Z\ IO'_G(Z) \quad \Leftarrow \quad G^-\ .$$

Thus in example 16, the partial correctness of Q' writes, for all u and v, as
$$\forall z\ \exists y\ \ append(u, v, y) \wedge length(y, z) \quad \Leftarrow \quad io'_G(u, v, z).$$
while the termination of Q' writes, for all u and v, as
$$\exists z\ \ io'_G(u, v, z).$$
Likewise in example 17, the partial correctness of the projected form writes, for all u and v, as
$$\forall y, z\ \exists k\ \ \times(y, u, k) \wedge \times(z, v, succ(k)) \quad \Leftarrow \quad gcd(u, v, succ(0)) \wedge io'_G(u, v, y, z).$$
while the termination property writes, for all u and v, as
$$\exists y, z\ \ io'_G(u, v, y, z) \quad \Leftarrow \quad gcd(u, v, succ(0)).$$

The two following propositions can be easily proved.

Proposition 24 *In the least Herbrand model of $Q \cup Q'$, we have:*
$$IO'_G(Z) \quad \Leftarrow \quad IO_G(Y, Z).$$

Proposition 25 *Suppose that Q is terminating w.r.t. G and $IO'_G(Z)$ is* functional *w.r.t. Z, i.e., in the least Herbrand model of Q', given two renamings Z_1 and Z_2 of Z*
$$Z_1 = Z_2 \quad \Leftarrow \quad IO'_G(Z_1) \wedge IO'_G(Z_2).$$
Then, in the least Herbrand model of $Q \cup Q'$, we have:
$$\forall Z\ \exists Y\ IO_G(Y, Z) \quad \Leftarrow \quad G^- \wedge IO'_G(Z).$$

From these propositions and definitions of partial correctness and termination, we have the following

Proposition 26 *Suppose that Q is partially correct and terminating w.r.t. G, and $IO'_G(Z)$ is functional w.r.t. Z. Then Q' is partially correct and terminating w.r.t. G.*

Thus, by proposition 26, the partial correctness and termination of Q' in example 16 follow from those of Q and from the functionality of $io'_G(u, v, z)$ w.r.t. z, i.e.:
$$z_1 = z_2 \quad \Leftarrow \quad io'_G(u, v, z_1) \wedge io'_G(u, v, z_2).$$
Likewise, the partial correctness and termination of the program defining $io'_G(u, v, y, z)$ in example 17 follow from partial correctness and termination of the program defining $io_G(u, v, k, y, z)$ and from the functionality of $io'_G(u, v, y, z)$ w.r.t. y, z, i.e.:
$$y_1 = y_2 \wedge z_1 = z_2 \quad \Leftarrow \quad io'_G(u, v, y_1, z_1) \wedge io'_G(u, v, y_2, z_2).$$

1.5

CONCLUSION

We have described a proof-extraction method for synthesizing logic programs, which is based on the system of extended execution. The synthesized program is guaranteed to terminate and to behave correctly with respect to the initial specification. As suggested in former examples, the synthesized program coincides in simple cases with the program obtained by the unfolding/folding method. This analogy has been further studied in [Bou91a, Bou91b]. The main defect of the method compared with unfolding/folding comes from the fact that our extracted logic program is only able to compute one correct output per input rather than all of them. (With this respect our program behaves as a deterministic function.) On the other hand, our method is able to synthesize programs that correspond to implicative specifications, and not only to conjunctive specifications as in unfolding/folding. Once a proof by extended execution and induction has been constructed, the synthesis of the logic program is purely mechanical. The main obstacle for extracting programs in a fully automatic way lies in the construction of the proof itself because proofs by induction heavily rely on heuristics and on the discovery of useful lemmas. A solution to this problem might be to implement refined principles of induction or to exploit methods of automatic generation of lemmas (see, e.g., [BvHSI90, DB89, Fri91]).

1.6

APPENDIX: PREDICATE DEFINITIONS

$append([], y, y).$
$append([p|x], y, [p|z]) \leftarrow append(x, y, z).$

$length([], 0).$
$length([p|x], succ(y)) \leftarrow length(x, y).$

$0 < succ(y).$
$succ(x) < succ(y) \leftarrow x < y.$

$+(0, y, y).$
$+(succ(x), y, succ(z)) \leftarrow +(x, y, z).$

$\times(0, y, 0).$
$\times(succ(x), y, z) \leftarrow \times(x, y, w), +(y, w, z).$

$nonemptylist([p|x]).$

$minimum([p], p).$
$minimum([p|x], p) \leftarrow minimum(x, p).$
$minimum([p|x], p) \leftarrow minimum(x, q), p < q.$
$minimum([p|x], q) \leftarrow minimum(x, q), q < p.$

$maximum([p], p).$
$maximum([p|x], p) \leftarrow maximum(x, p).$

$maximum([p|x], q) \leftarrow maximum(x, q), p < q.$
$maximum([p|x], p) \leftarrow maximum(x, q), q < p.$

$sum([], 0).$
$sum([p|w], y) \leftarrow sum(w, z), +(z, p, y).$

$squarelist([], 0).$
$squarelist([p|w], [q|y]) \leftarrow squarelist(w, y), \times(p, p, q).$

$fibonacci(0, succ(0)).$
$fibonacci(succ(0), succ(0)).$
$fibonacci(succ(succ(x)), y) \leftarrow fibonacci(succ(x), z), fibonacci(x, w), +(z, w, y).$

$length4([], [], [], [], 0).$
$length4([p|w], [q|x], [r|y], [s|z], succ(n)) \leftarrow length4(w, x, y, z, n).$

$dot([], [], 0).$
$dot([p|w], [q|x], succ(n), y) \leftarrow dot(w, x, n, z), \times(p, q, r), +(z, r, y).$

1.7

EXERCISES

1. Prove by extended execution the goal $G:\ length(y, ?z) \Leftarrow append(u, v, y)$. Extract the definition of $io_G(u, v, y, z)$ from the proof. Compare with the definition obtained in example 14.
2. Prove by extended execution the goal $G:\ (minimum(x, ?y) \wedge maximum(x, ?z)) \Leftarrow nonempty(x)$. Extract the definition of $io_G(x, y, z)$ from the proof.
3. Prove by extended execution the goal $G:\ squarelist(x, ?y) \wedge sum(?y, ?z)$. Extract the definition of $io_G(x, y, z)$ from the proof. To which argument(s) of io_G would you apply a projection?
4. Prove by extended execution the goal $G:\ fibonacci(x, ?y) \wedge fibonacci(succ(x), ?z)$. Extract the definition of $io_G(x, y, z)$ from the proof.
5. Prove by extended execution $G:\ (dot(w, x, n, ?a) \wedge dot(y, z, n, ?b) \wedge +(?a, ?b, ?c)) \Leftarrow length4(w, x, y, z, n)$. Extract the definition of $io_G(a, b, c, n, w, x, y, z)$ from the proof. To which argument(s) of io_G would you apply a projection?

1.8

*ACKNOWLEDGEMENT

I would like to thank C. Paulin-Mohring, L. Colson, A. Bouverot and H. Olsen for numerous fruitful discussions, and T. Sato and T. Kanamori for helpful comments on an earlier draft of this chapter. This work has been partially supported by ESPRIT

Project BRA 3020.

1.9

*REFERENCES

[Bou91a] A. Bouverot. *Comparaison entre la Transformation et l'Extraction de Programmes Logiques.* PhD thesis, (in French), Université Paris 7, March 1991.

[Bou91b] A. Bouverot. Extracting and Transforming Logic Programs. Technical Report LIENS-91-4, Ecole Normale Supérieure, Paris, March 1991.

[BSW90] A. Bundy, A. Smaill, and G. Wiggins. The Synthesis of Logic Programs from Inductive Proofs. In J.W. Lloyd, editor, *Symposium on Computational Logic*, Esprit Basic Research Series, pages 135–149. Springer-Verlag, Heildelberg, Germany, 1990.

[Bun88] A. Bundy. A Broader Interpretation of Logic in Logic Programming. In R.A. Kowalski and K.A. Bowen, editors, *Proc. Fifth International Conference and Symposium on Logic Programming*, Series in Logic Programming, pages 1624–1648, Seattle, USA, August 1988. The MIT Press.

[Bur69] R.M. Burstall. Proving Properties of Program by Structural Induction. *Computer J.*, 12(1):41–48, February 1969.

[BvHSI90] A. Bundy, F. van Harmelen, A. Smaill, and A. Ireland. Extensions to the Rippling-Out Tactic for Guiding Inductive Proofs. In M.E. Stickel, editor, *Proc. 10th International Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence 449, pages 132–146, Kaiserslautern, 1990. Springer-Verlag.

[CAea86] R.L. Constable, S.F. Allen, and H.M. Bromley et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice Hall, New Jersey, 1986.

[Cla78] K. Clark. Negation as Failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322. Plenum Press, New York, USA, 1978.

[Cla79] K.L. Clark. Predicate Logic as a Computational Formalism. Research Monograph 79/59, TOC, Imperial College, London, December 1979.

[CS77] K.L. Clark and S. Sickel. Predicate Logic: A Calculus for Deriving Programs. In *Proc. International Joint Conference on Artificial Intelligence*, pages 419–420, Cambridge (MA), August 1977.

[Day87] G. Dayantis. Logic Program Derivation for a Class of First Order Logic Relations. In J. McDermott, editor, *Proc. International Joint Conference on Artificial Intelligence*, pages 9–14, Milano, August 1987.

[DB89] Y. Deville and J. Burnay. Generalization and Program Schemata: A Step Towards Computer-Aided Construction of Logic Programs. In E.L. Lusk and R. Overbeek, editors, *Proc. North American Conference on Logic Programming*, Series in Logic Programming, pages 409–425, Cleveland, USA, October 1989. The MIT Press.

[Eri84] L-H. Eriksson. Synthesis of a Unification Algorithm in a Logic Programming Calculus. *Journal of Logic Programming*, 1(1):3–18, 1984.

[Fri88] L. Fribourg. Equivalence-Preserving Transformation of Inductive Properties of Logic Programs. In R.A. Kowalski and K.A. Bowen, editors, *Proc. Fifth International Conference and Symposium on Logic Programming*, Series in Logic Programming, pages 103–116, Seattle, USA, August 1988. The MIT Press.

[Fri91] L. Fribourg. Automatic Generation of Simplification Lemmas for Inductive Proofs. In V. Saraswat and K. Ueda, editors, *Proc. Fifth Symposium on Logic Programming*, Series in Logic Programming, pages 103–116, San Diego, USA, October 1991. The MIT Press.

[GWH91] I. Kraan G. Wiggins, A. Bundy and J. Hesketh. Synthesis and Transformation of Logic Programs through Constructive, Inductive Proofs. In K. Lau and T. Clement, editors, *Proc. LOPSTR'91*, Workshop in Computing Series, pages 27–45, Manchester, July 1991. Springer-Verlag.

[Hay86] S. Hayashi. PX: A System Extracting Programs from Proofs. In M. Wirsing, editor, *Proc. 3rd Working Conference on the Formal Description of Programming Concepts*, Ebberup, Danemark, August 1986. North-Holland.

[Hog81] C.J. Hogger. Derivation of Logic Programs. *Journal of ACM*, 28(2):372–392, 1981.

[HT79] A. Hansson and S.-A. Tarnlund. A Natural Programming Calculus. In *Proc. International Joint Conference on Artificial Intelligence*, pages 348–355, Tokyo, August 1979.

[Kan88] T. Kanamori. Soundness and Completeness of Extended Execution for Proving Properties of Prolog Programs. In K. Fuchi and M. Nivat, editors, *Programming of Future Generation Computers*, pages 259–281. Elsevier Science Publishers B.V., North-Holland, 1988.

[KF86] T. Kanamori and H. Fujita. Formulation of Induction Formulas in Verification of Prolog Programs. In J.H. Siekmann, editor, *Proc. 8th International Conference on Automated Deduction*, Lecture Notes in Computer Science 230, pages 281–299, Oxford, July 1986. Springer-Verlag.

[KS86] T. Kanamori and H. Seki. Verification of Prolog Programs Using an Extension of Execution. In E.Y. Shapiro, editor, *Proc. Third International Conference on Logic Programming*, volume 86 of *Lecture Notes in Computer Science*, pages 475–489, London, United Kingdom, July 1986. Springer-Verlag.

[ML79] P. Martin-Löf. Constructive Mathematics and Computer Programming. In North-Holland, editor, *Proc. 6th International Congress for Logic, Methodology and Philosophy of Science*, pages 153–175, Hanover, 1979. North-Holland.

[Moh86] C. Mohring. Algorithm development in the calculus of constructions. In A. Meyer, editor, *Proc. IEEE Symposium on Logic in Computer Science*, pages 84–91, Cambridge, MA, June 1986. North-Holland.

[MW80] Z. Manna and R. Waldinger. A Deductive Approach to Program Synthesis. *ACM Trans. Prog. Lang. Syst.*, 2(1):90–121, 1980.

[Nar89] D. Nardi. Formal Synthesis of a Unification Algorithm by the Deductive-Tableau Method. *Journal of Logic Programming*, 7(1):1–43, 1989.

[PP89] A. Pettorossi and M. Proietti. Decidability Results and Characterization of Strategies for the Development of Logic Programs. In G. Levi and M. Martelli, editors, *Proc. Sixth International Conference on Logic Programming*, Series in Logic Programming, pages 539–553, Lisbon, Portugal, July 1989. The MIT Press.

[Rog67] H.R. Rogers. *Theory of Recursive Functions and Effective Computability.* McGraw-Hill, New York, 1967.

[Sko67] T. Skolem. The foundations of elementary arithmetic established by means of the recursive mode of thought, without the use of apparent variables ranging over infinite domains. In J. van Heijenoort, editor, *From Frege to Gödel: a Source Book in Mathematical Logic 1879-1931*, pages 302–333. Harvard University Press, 1967.

[Tak87] Y. Takayama. Writing Programs as QJ Proof and Compiling into Prolog Programs. In S. Haridi, editor, *Proc. Fourth Symposium on Logic Programming*, pages 278–287, San Francisco, USA, August - September 1987. The IEEE Computer Society Press.

[TS84] H. Tamaki and T. Sato. Unfold/Fold Transformation of Logic Programs. In S. Tärnlund, editor, *Proc. Second International Conference on Logic Programming*, pages 127–138, Uppsala, Sweden, July 1984. Upsala University Press.