Fundamenta Informaticae XIV (1991) 339-343
IOS Press

ARITHMETIC CLASSIFICATION OF PERFECT MODELS OF STRATIFIED PROGRAMS

Krzysztof R. APT

Centre for Mathematics and Computer Sciences, P.O. Box 4079, 1009 AB Amsterdam, Netherlands
and
Department of Computer Sciences, University of Texas at Austin, Austin, Texas 78712-1188, USA

and

Howard A. BLAIR

School of Computer and Information Science, 4-116 CST, Syracuse University, Syracuse, NY 13244-4100, USA

Recursion-free programs

The following section completes the analysis of arithmetic complexity of perfect models and has been inadvertently omitted in the previous version of the paper.

We say that a general program P is *recursion-free* if in its dependency graph D_P there is no cycle. Clearly recursion-free programs form a subclass of stratified programs. Recursion-free programs form a very simple generalization of the class of *hierarchical* programs introduced in [C78]. Hierarchical programs satisfy an additional condition on variable occurrences in clauses that prevents *floundering*, i.e. a forced selection of a non-ground negative literal in an SLDNF- derivation. In this section we study the complexity of perfect models of recursion-free programs.

1. Hierarchical stratifications

We call a stratification

$$P = P_1 \cup \cdots \cup P_n$$

of a general program *hierarchical* if for $i = 1,...,n$ and for every relation symbol which occurs in a body of a general clause from P_i, its definition is contained within some P_j for $j<i$.

The following lemma shows that general programs admitting hierarchical stratifications and recursion-free programs coincide. It is in fact a special case of the well known fact that a finite relation can be topologically sorted iff it is acyclic. Therefore we omit the proof.

LEMMA 1: *A general program P is recursion-free iff there is a hierarchical stratification of P.* □

*) to the version of this paper that appeared in Fundamenta Informaticae 13, pp.1-18, 1990.

2. Completions of recursion-free programs
In the sequel we shall study comp(P), CLARK'S [C78] completion of a general program P. Its definition can be found in [Ll87]. comp(P) is a theory whose language is the first order language $L(P)$ of the general program P augmented by the equality relation symbol "$=$". Given a general program P we denote by $L(=)$ the language obtained by deleting from $L(P)$ all relation symbols and by adding "$=$" to it. From now on in presence of a general program P we denote $L(P)$ by L.

comp(P) is a set of formulas which consists of *free equality axioms*, which we denote by Eq, together with certain other formulas, about which we only need to know the following two properties.

PROPERTY 1: For every m-ary relation symbol q of L with the empty definition in P, the formula

$$\forall x_1 \ldots \forall x_m \neg q(x_1, \ldots, x_m)$$

is in comp (P).

PROPERTY 2: For every m-ary relation symbol q of L with the non-empty definition in P, there is in comp (P) a formula of the form

$$q(x_1, \ldots, x_m) \leftrightarrow \psi_q$$

such that every relation symbol occurring in ψ_q, other than "$=$", occurs in the definition of q in P.

The equality axioms in Eq are the usual axioms of first-order logic with equality that say that $=$ is a congruence, together with axioms that say that the function symbols of L, including the 0-ary ones, denote one-one functions with disjoint ranges, and axioms which say that all functions definable by composition from the given function symbols have no fixed points. As pointed out by KUNEN [K87], these are the axioms required for justifying the soundness of both success and failure of unification.

In the proof of the next lemma we shall need the following result from mathematical logic (see [Sh67] p. 34).

THEOREM 2: *(Equivalence Theorem). Let* T *be a theory and* ϕ *a formula. Suppose that* ϕ' *is obtained from* ϕ *by replacing some, possibly all occurrences of subformulas* $\psi_1, \ldots, \psi_n$ *by* $\psi'_1, \ldots, \psi'_n$ *respectively. Then if for* $i = 1, \ldots, n$,

$$\mathrm{T} \vdash \psi_i \leftrightarrow \psi'_i,$$

then

$$\mathrm{T} \vdash \phi \leftrightarrow \phi'.$$

□

Here replacing involves an appropriate renaming of variables performed in order to avoid variable clashes.

LEMMA 3: *Let P be a recursion-free program. For every atom A of L there exists a formula* ϕ_A *of* $L(=)$ *all of whose free variables occur in A such that*

$$\mathrm{comp}(P) \vdash A \leftrightarrow \phi_A. \tag{1}$$

PROOF. Let $P = P_1 \cup \cdots \cup P_n$ be a hierarchical stratification of P whose existence is guaranteed by lemma 1. We define a mapping *height* from relation symbols of L

into $\{0, 1, ..., n\}$ as follows.

Let r be a relation symbol of L whose definition within P is empty. Then we put height$(r)=0$. Otherwise we put height$(r)=i$ iff the definition of r is contained in P_i.

Suppose that $A \equiv r(t_1, \ldots, t_m)$ for a relation symbol r and terms $t_1, \ldots, t_m$. We prove the lemma by induction on the height of r. If height$(r)=0$, then by property 1

$$\text{comp}(P) \vdash A \leftrightarrow \mathit{false},$$

so we can take **false** for ϕ_A. We can regard **false** as an abbreviation of $\neg\forall x(x=x)$. If height $(r)=1$, then by property 2 ψ_r is a formula from $L(=)$ with free variables $x_1, \ldots, x_m$. Let ψ'_r stand for $\psi_r\{x_1/t_1, \ldots, x_m/t_m\}$. Then ψ'_r is a formula from $L(=)$ all of whose free variables occur in A such that

$$\text{comp}(P) \vdash A \leftrightarrow \psi'_r, \qquad (2)$$

so we can take ψ'_r for ϕ_A.

Assume now that the claim holds for all relation symbols with height $<k$ and suppose that height$(r)=k$. By property 2, (2) holds, but where, instead of ψ'_r being a formula of $L(=)$, every relation symbol q occurring in ψ'_r and different from "$=$" occurs in the definition of r in P. But the stratification

$$P = P_1 \cup \cdots \cup P_n$$

of P is hierarchical, so every such relation symbol q is of height $<k$. Thus by the induction hypothesis, for every atom B occurring in ψ'_r and whose relation symbol differs from "$=$", there exists a formula ϕ_B if $L(=)$ all of whose free variables occur in B such that

$$\text{comp}(P) \vdash B \leftrightarrow \phi_B. \qquad (3)$$

Now, replace each occurrence of such an atom B in ψ'_r by ϕ_B and call the resulting formula ϕ_A. Note that ϕ_A is a formula of $L(=)$ and that all its free variables appear in A. Now by theorem 2 we get (1) by virtue of (2) and (3).

□

COROLLARY 4: *Let P be a recursion-free program. For every formula ϕ of L there exists a formula ψ of $L(=)$ all of whose free variables occur as free variables in ϕ and such that*

$$\text{comp}(P) \vdash \phi \leftrightarrow \psi_\phi.$$

PROOF. By lemma 3 for every atom A occurring in ϕ there exists a formula ϕ_A of $L(=)$ all of whose free variables occur in A and such that (1) holds. Now, replace each occurrence of an atom A in ϕ by ϕ_A and call the resulting formula ψ_ϕ. Then ψ_ϕ is a formula of $L(=)$ all of whose free variables occur as free variables in ϕ. By theorem 2 we now get the desired conclusion by virtue of (1).

□

3. *Domain closure axiom*

In the sequel we shall refer to a number of basic concepts from mathematical logic which we now briefly recall.

By a *closed formula* we mean a formula without free variables. A theory T is called *complete* if for all closed formulas ϕ either $T \vdash \phi$ or $T \vdash \neg\phi$. A theory T is called *consistent* if for no formula ϕ both $T \vdash \phi$ and $T \vdash \neg\phi$. Finally, a theory T is called *decidable* if (after the standard encoding) the set $\{\phi : T \vdash \phi\}$ is recursive.

Let L be a first order language with finitely many function symbols and constants.

By *DCA* (the *d*omain *c*losure *a*xiom) we mean the following first order formula of L:

$$\forall x \bigvee_f \exists y_1 \ldots \exists y_n (x = f(y_1, \ldots, y_n)),$$

where n is the number of arguments of f, and is 0 if f is a constant. Thanks to the restriction on L, *DCA* is indeed a first order formula. For example, if L contains one constant a, one unary function symbol f and one binary function symbol g, then *DCA* can be taken as

$$\forall x(x = a \vee \exists y(x = f(y) \vee \exists y_1 \exists y_2(x = g(y_1, y_2)))).$$

We now need the following result due to MAHER [M88].

THEOREM 5: *Let L be a first order language with = and with finitely many function symbols and constants but at least with one constant. Then* $Eq \cup \{DCA\}$ *is a complete and decidable theory.* □

Note that the Herbrand base corresponding to the language L augmented by "=" is a model of $Eq \cup \{DCA\}$, so $Eq \cup \{DCA\}$ is also a consistent theory.

Recall that we originally assumed that each general program P contains at least one constant and one function symbol. So we can apply the above theorem here. We can now prove the main result of this section.

THEOREM 6: *Let P be a recursion-free program. Then* $comp(P) \cup \{DCA\}$ *is a complete and decidable theory.*

PROOF. Let ϕ be a closed formula of the language $L(P)$ augmented by =. Then the formula ψ_ϕ from corollary 4 is closed, as well. By corollary 4

$$\text{comp}(P) \cup \{DCA\} \vdash \phi \textit{ iff } \text{comp}(P) \cup \{DCA\} \vdash \psi_\phi .$$

Moreover by theorem 5

$$\text{comp}(P) \cup \{DCA\} \vdash \psi_\phi \textit{ iff } Eq \cup \{DCA\} \vdash \psi_\phi ,$$

since comp(P) ∪ {*DCA*} is consistent. Combining these two equivalences we get

$$\text{comp}(P) \cup \{DCA\} \vdash \phi \textit{ iff } Eq \cup \{DCA\} \vdash \psi_\phi . \tag{4}$$

But by the form of ψ_ϕ we have that $\psi_{\neg\phi}$ is identical to $\neg\psi_\phi$, so since $\neg\phi$ is a closed formula, as well,

$$\text{comp}(P) \cup \{DCA\} \vdash \neg\phi \textit{ iff } Eq \cup \{DCA\} \vdash \neg\psi_\phi . \tag{5}$$

Now by virtue of theorem 5, (4) implies that comp(P) ∪ {*DCA*} is decidable and (4) and (5) imply that comp(P) ∪ {*DCA*} is complete.

□

COROLLARY 7: *Let P be a recursion-free program. Then for every ground atom A*

$$A \in M_P \textit{ iff } \text{comp}(P) \cup \{DCA\} \vdash A .$$

PROOF. M_P is a model of comp(P) ∪ {*DCA*}. Thus $A \in M_P$ implies that comp(P) ∪ {*DCA*} ⊢ ¬A does not hold which by theorem 6 implies comp(P) ∪ {*DCA*} ⊢ A. Also, $A \notin M_P$ implies that comp (P) ∪ {*DCA*} ⊢ A does not hold.

□

We can obtain the desired conclusion.

COROLLARY 8: *Let P be a recursion-free program. Then M_P is recursive.*

PROOF. By corollary 7 and theorem 6.

□

4. Discussion

It is straightforward to define the completion of a program, hence Eq, and in the case of stratified programs, the standard model M_P, with respect to a language L with possibly infinitely many function and relation symbols, where L is any particular extension of the smallest language $L(P)$ in which the clauses of P can be expressed. It is easy to see that M_P is in fact dependent on the underlying language L. The point of view where all programs are taken as sets of clauses over the same denumerable (effectively presented) language L is extensively discussed by Maher [M88a].

If comp(P) and M_P are defined in this way with respect to L, the fundamental results that M_P is independent of the stratification of P, and that M_P is a model of comp(P) continue to hold.

If the set of function symbols of L is infinite (and the sets of function and relation symbols of L are suitably effectively presented), then the theorem and corollaries of section 3 continue to hold, provided *DCA* is deleted from their statements and proofs. This holds because for such a language L, the equality theory Eq, (without a domain closure axiom, which would require an infinite disjunction to express) is complete and decidable, (cf. [K87]).

REFERENCES

[C78] K.L. CLARK, Negation as Failure, in: *Logic and Databases,* (H. Gallaire and J. Minker, eds.), Plenum Press, New York, 1978, pp. 293-322.

[Ll87] J.W. LLOYD, *Foundations of Logic Programming, 2nd edition,* Springer-Verlag, 1987.

[K87] K. KUNEN, *Negation in Logic Programming.* J. of Logic Programming, Vol. 4, 1987, pp. 289-308.

[M88] M.J. MAHER, Complete Axiomatizations of the Algebras of Finite, Rational and Infinite Trees, in: *Proc. of the 3rd Logics in Computer Science Conference,* 1988, pp. 348-357.

[M88a] M.J. MAHER, *Equivalences of Logic Programs,* in: Foundations of Deductive Databases and Logic Programming, (Jack Minker, ed.) Morgan-Kaufmann, Los Altos, CA. 1988, pp. 627-658.