Logical Constraints and Logic Programming [1]

V. Wiktor Marek, [2] Anil Nerode, [3] and Jeffrey B. Remmel [4]

Abstract

In this note we will investigate a form of logic programming with constraints. The constraints that we consider will not be restricted to statements on real numbers as in CLP($\mathbf{R}$), see [15]. Instead our constraints will be arbitrary *global* constraints. The basic idea is that the applicability of a given rule is not predicated on the fact that individual variables satisfy certain constraints, but rather on the fact that the least model of the set rules that are ultimately applicable satisfy the constraint of the rule. Thus the role of clauses will be slightly different than in the usual Logic Programming with constraints. In fact, the paradigm we present is closely related to stable model semantics of general logic programming [13]. We will define the notion of a constraint model of our constraint logic program and show that stable models of logic programs as well as the supported models of logic programs are just special cases of constraint models of constraint logic programs.

Our definition of constraint logic programs and constraint models will be quite general. Indeed, in general definition, the constraint of a clause will not be restricted to be of a certain form or even to be expressible in the underlying language of the logic program. This feature is useful for certain applications in hybrid control systems and database applications that we have in mind. However for the most part in this paper, we focus on the properties of constraint programs and constraint models in the simplest case where the constraints are expressible in the

[1] A short version of this paper has been published in the Proceedings of the Third International Conference on Logic Programming and Nonmonotonic Reasoning, Springer Lecture Notes in Computer Science 927, 1995

[2] Department of Computer Science, University of Kentucky, Lexington, KY 40506, Research partially supported by NSF grant IRI-9400568. E-mail: *marek@cs.engr.uky.edu*

[3] Mathematical Sciences Institute, Cornell University, Ithaca, NY 14853, Research supported by US ARO contract DAAL03-91-C-0027. E-mail: *anil@math.cornell.edu*

[4] Department of Mathematics, University of California, La Jolla, CA 92093. Research partially supported by NSF grant DMS-9306427. E-mail: *remmel@kleene.ucsd.edu*

language of underlying program. However even in this case, we will show that some new phenomenon can occur. Because stable models are special cases of constraint models, constraint models have that property that there may be zero or many constraint models for a given constraint program and that the constraint models may be quite complex. Nevertheless we will show in the case of propositional logic where constraints are positive formulas, there always exists a least constraint model and it can be computed in a polynomial time.

1 Introduction

Constraint Logic Programming received a good deal of attention since its introduction in the seminal paper of Jaffar and Lassez in 1986 [15]. The crucial observation of Jaffar and Lassez is that not all atoms in the body of a logic program play the same role in the evaluation process. They noted that it is actually beneficial to separate some goals, such as goals of the form $f(X_1, \ldots, X_n) = a$ and goals of the form $f(X_1, \ldots, X_m) \leq a$, into the separate parts which can be evaluated or solved using specialized resources of the underlying computer system. Thus a part of the goals act as constraints in the space of possible solutions. One can interpret this phenomenon with a slightly shifted emphasis. Namely we can think of the constraints as conditions that need to be satisfied by (some) parameters before we start to evaluate the remaining goals in the body of the clause. Thus the constraint controls the applicability of the clause, that is, if it fails, then the rule cannot fire. It is this point of view that we shall generalize in this paper. Our idea is that constraints can be quite arbitrary and that their role is to control the applicability of the rule. Nevertheless we shall see that in certain special cases our generalization still fits quite naturally with the idea of solving the constraints via "specialized equipment".

The constraints in Constraint Logic Programming are evaluated in some fixed domain. This is generally some fixed relational system. In the specific schemes proposed for constraint logic programming, it can be the set of reals with some specific operations and/or relations, as in the original $CLP(R)$ or a Boolean Algebra as in CHIP [11] or the set of finite or infinite trees as in [10]. Here we will not make that type of assumption. That is, the

domain where the constraints are evaluated is **not** necessarily fixed upfront. This decision makes the resulting theory much more flexible. In fact all the previously mentioned schemes can be embedded in our scheme. It should be noted that already Jaffar and Lassez showed that the constraint logic programming scheme $CLP(R)$ can be embedded in logic programming. The gain is not in the conceptual level but in the effectiveness of processing. By contrast, our notions of constraint logic programs and constraint models is a genuine extension of logic programming and logic programming with negation as failure. We will show that appropriate translations allow us to express both the supported semantics of logic programs [9] and stable semantics for such programs (therefore also perfect semantics [3] as well) as the particular cases of our scheme. Nevertheless the relaxation of the conditions on the constraints still leads to a coherent and clean theory of what we call below constraint models of constraint programs.

The novel feature of the theory proposed below is that the constraints are supposed to be true in the model consisting of the atoms computed in the process. That is, we allow for an *a posteriori* justification of the rules applied in the process of computation. At first glance a different phenomenon occurs in constraint logic programming, namely the constraints are applied *a priori*. However we shall see below that both approaches essentially coincide in the case of upward preserved constraints. These are constraints which have the property that they are preserved as we go from a smaller to a bigger model, see section 4. This is precisely what happens in constraint logic programming. Once a constraint is computed, it is maintained. Indeed, in constraint logic programming, once we find the multidimensional region containing the solutions of the constraint, the subsequent steps can only refine this description, the solutions must come from that region. We will see, however, that even in the case of upward preserving constraints, we get a scheme more powerful than ordinary logic programming. We shall show the least constraint model always exists and can be computed via the usual construction of the least fixed point of a monotone operator but that the closure ordinal for the least constraint model can be strictly bigger than ω. That is, the operators corresponding to such programs are monotone but are not always compact. Thus there will be the least and largest fixpoints for such operators. The least fixpoint will be the least constraint model. The largest fixpoint will be the largest constraint model but of a transformed

program. The fact that both the upwards and downwards iterations of the operator are transfinite make our class of programs less irregular in a certain sense than the usual operators for Horn logic programming. Recall the fact that in Horn logic programming the least fixpoint is reached in at most ω steps whereas the greatest fixed point can take up to ω_1^{CK} iterations, see Blair [6].

The constraints we allow may be very diverse and complex formulas. They may be formulas in the underlying language of the program or they may be formulas in second order or even infinitary logic. An example of this type of constraint is the parity example where constraint is a statement about the parity of the (finite) putative model of the program. This type of the constraint is a formula of infinitary logic if the Herbrand base is infinite. What is important is that there is a method to check if the constraint holds relative to a possible model. Thus we will assume that we have a *satisfaction relation* between the subsets of Herbrand base and formulas that can be used in constraints. Of course, $CLP(R)$ and similar schemes make the same type of assumptions.

Our motivation for allowing more general constraints originally came from the certain applications in control theory of real-time systems. The basic idea is that we sample the plant state at discrete intervals of time, $\Delta, 2\Delta, \ldots$. Based on the plant measurements, a logic program will compute the control to be used by the plant for the next Δ seconds so that the plant state will be guaranteed to meet certain required specifications. One possible way for such logic programs to operate is that the set of rules with which we compute at any give time $n\Delta$ is a function of the observations of the state of the plant at time $n\Delta$. In this fashion, we can view the plant state at time $n\Delta$ as determining which constraints of the rules of the logic program are satisfied and hence which rules can fire at time $n\Delta$. In such a situation, we cannot expect that we will have the constraints which are upward preserving or even that the constraints should necessarily be in the same language as the underlying language of the program. Therefore it will be definitely necessary to step out of the current constraint logic programming paradigm. Another application of the same type is to have a logic program controlling inventory via a deductive database. In this case, we may want to change the rules of the deductive database as a function of outside demand. Once again one could view the satisfaction of the constraints as depending on the database for the

inventory and the set of orders that come in at a given time. In this way one can vary the set of applicable rules as function of the current inventory and demand.

The paper is organized as follows. We investigate the basic properties of constraint programs in Section 2. Propositional programs with constraints are studied in Section 3. We show basic properties of such programs in our setting. It turns out that both stable and supported models of programs can be modelled as constraint programs. In Section 4 we discuss programs with arbitrary constraints. We show that programs with upward-preserving constraints have always a least constraint model, although the closure ordinal of the construction is no longer ω. Section 5 contains a proof theoretic characterization of constraint models. In Section 6 we show how Default Logic can be simulated by programs with constraints. Section 7 contains a technique for construction of constraint models. This technique is complete for both upward-preserving and downward-preserving constraints. We show that the alternating fixpoint technique for constructing a three-valued model can be used for constraint programs with downward-preserving constraints.

2 Preliminaries

First, we shall introduce our syntax. The clauses we consider are of the form:

$$C = p \leftarrow q_1, \ldots, q_m : \Phi$$

Here $p, q_1, \ldots, q_m$ are (ground) atoms. Φ is a formula of some formal language. In the most general case we do not put restrictions on what Φ will be. The (Horn) clause $p \leftarrow q_1, \ldots, q_m$ will be denoted by H_C. The formula Φ is called the constraint of C and is denoted by Φ_C.

The idea behind this scheme is the following: Given a set of atoms, i.e. a subset of Herbrand base, M, we consider the formulas Φ_C for clauses C of the program. Some of these formulas are satisfied by the structure determined by M. This satisfaction is checked as follows. In the case of predicate logic, M determines a relational system $\mathcal{M}_M$ which is used for testing if Φ is true. Notice that, in principle, Φ can be any sort of formula. For instance, Φ may be a formula of second order logic or of some infinitary logic. In the case of

propositional logic, M determines a valuation that can be used to test Φ is true. Then we let A_M be the set of all clauses C such that $\mathcal{M}_M \models \Phi_C$. If we drop the constraints from the clauses in A_m, we get a set of Horn clauses H_C, i.e. a Horn program, which we denote by P_M. Then P_M has a least Herbrand model N_M and we say that M is a *constraint* model for P if $N_M = M$.

Although the use of formulas from outside of propositional logic or predicate logic may not seem very natural, it is easy to construct natural examples where the constraints naturally go beyond the expressibility in these logics. Indeed there has been significant work in deductive database theory on extensions of Datalog where one extends the underlying language by adding constructs like a transitive closure operator, [1]. Here is a simple example involving parity constraints.

Example 2.1 We consider a propositional program P with constraints on the parity of the target model. In a infinite propositional language, the formula expressing the fact that the number of the atoms in the model is even (or odd) is infinitary formula. Let P be the following program with logical constraints:

$p \leftarrow : \top$
$s \leftarrow q \ : \top$
$q \leftarrow p \ :$ "parity odd"
$r \leftarrow p \ :$ "parity even"

Here the formula "parity odd" is a formula saying that the number of atoms true in the model is odd. Likewise "parity even" is the infinitary formula saying that the number of atoms true in the model is even. In a model with *finite* number of true atoms, one of these formulas will be true. Of course, in the model with infinite number of true atoms both of these formulas will be false. Notice that these formulas are, in fact, infinite alternatives of infinite conjunctions.

Returning to our example, it should be clear that P has two constraint models. The first one $\{p, q, s\}$ has parity one, and the other one $\{p, r\}$ has parity zero.
Let us add now to this program P additional clause:

$$C = w \leftarrow r \ : \top$$

Then the program $P \cup \{C\}$ has just one constraint model, $\{p, q, s\}$. The reason the other model disappeared is that the presence of an (unconditional) clause C allows us to derive w once r has been derived. But the derivation of w changes the parity of the constructed set of atoms and hence invalidates the derivation of r which in turn invalidates the derivation of w as well.

In the next section we shall look at the simplest case where the underlying logic is propositional logic and the constraints Φ_C are also formulas of the propositional logic.

3 Propositional programs with constraints

First of all notice that if all constraints are equal to $\top$ then our construction of a constraint model reduces to the the least Herbrand model as in ordinary Horn logic programming. Thus in that case, the constraint model coincides in this case with the usual intended model of the program, namely, the least model.

Next, consider the case when the formulas Φ are of the form:

$$\neg r_1 \wedge \ldots \wedge \neg r_n$$

That is, the constraints are goals. It should be noted here that this type of constraint is frequently used within Artificial Intelligence. In this case constraint models are just stable models of an associated general logic program. That is, given a constraint clause with the constraint in the form of a goal

$$p \leftarrow q_1, \ldots, q_m : \neg r_1 \wedge \ldots \wedge \neg r_n,$$

assign to it a general clause

$$C' = p \leftarrow q_1, \ldots, q_m, \neg r_1, \ldots, \neg r_n$$

Then set $P' = \{C' : C \in P\}$. We then have this proposition

Proposition 3.1 *If P is a constraint program whose clauses have constraints in the form of a goal, then for every $M \subseteq H$, M is a constraint model of P if and only if M is a stable model of P'.*

Next, notice that the clauses with equivalent constraint have the same effect. Specifically we have the following proposition

Proposition 3.2 *If $\models \Phi \equiv \Psi$ and P is a constraint program such that for some clause C in P, $\Phi_C = \Phi$ and $\bar{P}$ a new program which results from P by substituting Ψ for Φ in one of places where it occurs as a constraint, then P and $\bar{P}$ have precisely the same constraint models.*

Next, consider the case when a formula $\Phi \equiv \Psi_1 \vee \ldots \vee \Psi_k$. Since changing Φ to $\Psi_1 \vee \ldots \vee \Psi_k$ does not change the class of constraint models, we can assume that $\Phi = \Psi_1 \vee \ldots \vee \Psi_k$.

Given a clause $C = p \leftarrow q_1, \ldots, q_m : \Phi$ let

$$C'' = \{p \leftarrow q_1, \ldots, q_m : \Psi_1, \ldots, p \leftarrow q_1, \ldots, q_m : \Psi_k\}$$

Then given a program P, define

$$P'' = \bigcup_{C \in p} C''$$

We have then

Proposition 3.3 *Let $M \subseteq H$. Then M is a constraint model of P if and only if M is a constraint model of P''*

Proof: We show that for all M, $P_M = P''_M$. This, of course implies the theorem.

Note that if $p \leftarrow q_1, \ldots, q_m \in P_M$, then for some $\Psi_1 \vee \ldots \vee \Psi_k$, $p \leftarrow q_1, \ldots, q_m : \Psi_1 \vee \ldots \vee \Psi_k \in P$, $M \models \Psi_1 \vee \ldots \vee \Psi_k$. But then for some $j \leq k$, $M \models \Psi_j$ and hence $p \leftarrow q_1, \ldots, q_m \in P''_M$. Thus $P_M \subseteq P''_M$. Vice versa, if $p \leftarrow q_1, \ldots, q_m \in P''_M$, then for some $\Psi_1 \vee \ldots \vee \Psi_k$, $p \leftarrow q_1, \ldots, q_m : \Psi_1 \vee \ldots \vee \Psi_k \in P$, $M \models \Psi_j$ for some $j \leq k$. But then $M \models \Psi_1 \vee \ldots \vee \Psi_k$, thus $p \leftarrow q_1, \ldots, q_m \in P_M$. Hence $P''_M \subseteq P_M$. □

Now call Φ purely negative if Φ is of the form $\Psi_1 \vee \ldots \vee \Psi_k$ and each Ψ_j is a goal. Propositions 3.1 and 3.3 imply the following

Proposition 3.4 *If each constraint in P is purely negative, then M is a constraint model of P if and only if M is a stable model of $(P'')'$.*

Since stable models of a general logic program form an antichain (i.e. are inclusion incompatible), the same holds for programs with purely negative constraints.

The inclusion-incompatibility of constraint models does not hold for arbitrary constraints. To this end consider this example:

Example 3.1 Let $P = \{p \leftarrow\!: p \wedge q, q \leftarrow\!: p \wedge q\}$. Then it is easy to see that both $\emptyset$ and $\{p, q\}$ are constraint models of P.

An analysis of example 3.1 lead us to the realization that the supported models of general logic programs can easily be described by means of a suitable transformation into programs with constraints. To this end let

$$C = p \leftarrow q_1, \ldots, q_m, \neg r_1, \ldots, \neg r_n$$

be a general logic program clause and assign to C the following constraint clause

$$C''' = p \leftarrow\!: q_1 \wedge \ldots \wedge q_m \wedge \neg r_1 \wedge \ldots \wedge \neg r_n$$

Then set $P''' = \{C''' : C \in \mathit{ground}(P)\}$. We have then

Proposition 3.5 *Let P be a general logic program. Then M is a supported model of P if and only if M is a constraint model of P'''.*

We consider another very natural case, namely when all constraints are purely positive. Here a formula is said to be purely positive if it is built from propositional letters using only conjunctions and alternatives. In this case constraint programs reduce (at a cost) to usual Horn programs. Here is how is it done. First we notice that we can assume that our constraints are in disjunctive normal form (Proposition 3.2) and then we notice that the

constraint distribute with respect to alternative (this fact has been used in the proof of Proposition 3.3). Then we execute the transformation in which the constraints which are now conjunctions of atoms are put into the bodies of their respective clauses. This is precisely how Jaffar and Lassez reduce $CLP(R)$ to the usual logic programming. We leave to the reader to show that the constraint model of the original program coincides with the least model of thus transformed program.

We conclude this Section with the complexity considerations for constraint programs which have propositional constraints. The existence problem for constraint models of such programs are, generally, on the first level of the polynomial hierarchy. Specifically, we can use Proposition 3.4 to prove the following:

Proposition 3.6 *The problem of existence of constraint model of a constraint program with propositional constraints is NP-complete.*

Proof: Since general propositional programs can be polynomially (in fact linearly) transformed to constraint programs so that stable models become precisely stable models (Proposition 3.4), the problem is NP-hard. On the other hand, once we take a guess, reduction and recomputation can certainly be made in polinomial time. Hence our problem is in NP. □

The result stated in Proposition 3.6 has been significantly extended by Pollet and Remmel [19]. and we state them here for completeness of the presentation.

Recall that a boolean quantified formula is an expression of the form

$$B = \exists \vec{x_1} \forall \vec{x_2} \ldots Q \vec{x_n} \Phi(\vec{a}, \vec{x_1}, \ldots, \vec{x_n})$$

or

$$B = \forall \vec{x_1} \exists \vec{x_2} \ldots Q \vec{x_n} \Phi(\vec{a}, \vec{x_1}, \ldots, \vec{x_n})$$

where Φ a Boolean term.

A program with boolean quantified constraints is a collection of constraint clauses where the constraints are boolean quantified formulas. $LPBQC_i$ is the collection of programs with boolean quantified constraints having at most

i alternations of quantifiers, In this setting general logic programs are precisely $LPBQC_0$ programs with constraints being the conjunctions of negated atoms. That is, the stable semantics for logic programs is obtained from constrained semantics for $LPBQC_0$ programs with constraints being conjunctions of negated atoms via simple linear transformation described above. Similarly, supported semantics was obtained also form $LPBQC_0$ with another transformation.

Recall now that by the classical result of Mayer and Stockmeyer [18] the satisfaction relation for the existential quantified boolean formulas with i alternations of quantifiers, $QSAT_i$ is a Σ^P_{i+1}-complete problem. We have then

Proposition 3.7 *([19]) The set of finite programs in $LPBQC_i$ that possess a constraint model forms a Σ^P_{i+1}-complete set.*

4 Arbitrary constraints

We turn our attention to constraints programs with constraints possessing some preservation properties. The preservation property in question is being preserved upwards.

Definition 4.1 A formula Φ is preserved upwards if whenever $M \subseteq N \subseteq H$ and $M \models \Phi$, then $N \models \Phi$.

The nature of the formula Φ is immaterial here, it can be a propositional formula, formula of the predicate calculus or even a formula of higher-order predicate calculus. What is important is that we have a satisfaction relation $\models$ which is used to check when a model satisfies a constraint.

Let Φ be a formula of propositional calculus built out of propositional variables by means of conjunction and alternative only. Then clearly Φ is preserved upwards. But in predicate calculus formulas built by use of existential quantifiers in addition to conjunction and alternative are also preserved upwards. The same happens in case of higher-order existential quantifiers.

Since constraint models of programs with purely negative constraints are stable models of an appropriate translation, it follows easily that some constraint programs have no constraint models. We show that for constraint program in which all constraints are preserved upwards there always exists a least constraint model.

Theorem 4.2 *Let P be a constraint program with upward preserved constraints. Then P has the least constraint model.*

Proof: Construct a subset of the Herbrand base by means of an inductive process as follows.

Let P_0 be the subset of P consisting of the clauses C for which $\Phi_C = \top$. The clauses of this subprogram are applied unconditionally. Program P_0 can be identified with the Horn program arising from P_0 by elimination of constraints altogether. Thus P_0 possesses the least model, M_0.

Now, assume that for some ordinal γ programs P_ξ and models M_ξ has been defined for all $\xi < \gamma$. Form an auxiliary structure $N_\gamma = \bigcup_{\xi<\gamma} M_\xi$. Now define P_γ as the set of those $C \in P$ such that $N_\gamma \models \Phi_C$. Then, strip P_γ of constraints. Thus, $P_\gamma = (P)_{N_\gamma}$. The resulting program is a Horn program and so it possesses the least model. Call this model M_γ.

Finally, set:

$$M = \bigcup_{\gamma \in Ord} M_\gamma$$

We claim that M is a constraint model of P. To prove our claim, we must first prove that prove that

$$P_M = \bigcup_{\gamma \in Ord} (P)_{M_\gamma}$$

Indeed, if a clause $p \leftarrow q_1, \ldots, q_m$ belongs to $\bigcup_{\gamma \in Ord}(P)_{M_\gamma}$ then for some Φ and $\gamma \in Ord$, $C = p \leftarrow q_1, \ldots, q_m : \Phi$, $C \in P$, $M_\gamma \models \Phi$. But Φ is preserving upwards, so $M \models \Phi$. Therefore $p \leftarrow q_1, \ldots, q_m$ belongs to P_M.

Conversely, if $p \leftarrow q_1, \ldots, q_m$ belongs to P_M, then for some Φ such that $M \models \Phi$, $p \leftarrow q_1, \ldots, q_m : \Phi$ is in P. But by the cardinality argument, for

some γ, $M_\gamma = M_{\gamma+1}$. Then $M\gamma = M$ and so the other inclusion is also established.

We are now in a position to prove that M is the least constraint model of P. First, let us see that M is indeed a constraint model of P_M. So, let $p \leftarrow q_1, \ldots, q_m$ belong to P_M and all $q_1, \ldots, q_m$ belong to M. Then for some γ, $p \leftarrow q_1, \ldots, q_m$ belongs to P_γ. Also there is δ such that $q_1, \ldots, q_m$ belong to M_δ. Let $\xi = \max(\gamma, \delta)$. Then $p \in M_{\delta+1}$. Thus $p \in M$.

Next, let R be any constraint model of P. By induction on γ we prove that $M_\gamma \subseteq R$. So, let $\gamma = 0$. Since $R \models \top$, $(P_0)_R \subseteq P_R$. Therefore R must contain M_0.

In the inductive step, we proceed similarly. The inductive assumption implies that $N_\gamma \subseteq R$. Since all the constraints preserve upwards, it follows that $P_\gamma \subseteq P_R$. This, in turn, implies that $M_\gamma \subseteq R$. □

As noticed above, when the constraints are propositional positive formulas, they certainly are preserved upwards. This means that for constraint programs with positive propositional constraints there always exists the least constraint model. This last result can be obtained independently via the simpler proof outlined at the end of section 3. Moreover, it can be proven that in that case the length of the hierarchy we construct is at most ω. In addition, if the program P is finite, the model M can be constructed in polynomial time.

When the constraints are formulas appearing in constraint part of the clauses are predicate formulas we face different choices of the satisfaction relations which, in turn, imply different closure conditions on the upward preserving formulas. This, in turn results in different closure ordinals of the resulting constraint model as constructed in Theorem 4.2. Among several subtle points here let us mention the issue of the set of atoms under consideration. We have this choice here: Are we taking into account only the terms actually appearing in the grounding of the program, or do we presuppose some Herbrand preinterpretation (set of constants and a set of function symbols) and a certain collection of relational symbols for the formation of ground atoms? We will look at this second case right now. That is we fix a Herbrand universe U and we fix a set of predicate symbols (together with their arities). In this fashion we get a fixed Herbrand base H. Subsets M of

H stand in one-to-one correspondence with Herbrand structures $\mathcal{M}_M$ with the universe U. The satisfaction relation $\models$ is the usual satisfaction relation for such models. We shall refer to Herbrand preinterpretation (to use the terminology of [2]).

Proposition 4.3 *p.herb Let a Herbrand preinterpretation U be given, and let $\models$ be the satisfaction relation between subsets of the Herbrand base H and the formulas of $\mathcal{L}$ be defined by $S \models \varphi$ if $\mathcal{M}_M \models \varphi$. Then the class of positive formulas (that is closure of atomic formulas under conjunction, alternative and both existential and universal quantifications) is upward-preserving.*

Proof: Although a direct proof by induction on the length of such formulas is possible, let us mention a more abstract argument showing what is happening here. The reason why the preservation holds is tha whenever $M_1 \subseteq M_2 \subseteq H$ then the identity is a homomorphism of $\mathcal{M}_{M_1}$ onto $\mathcal{M}_{M_2}$. Therefore the Lyndon homomorphism theorem ([8]) is applicable and the result follows. □

We will see now that indeed, with the arbitrary positive constraints as defined above the construction of the least constraint model may, indeed taka more than ω steps. In our next example we examine a program with positive constraints requiring $\omega + 1$ steps for the construction of the least constraint model.

Example 4.1 Let P be this program:

$p(0) \leftarrow : \top$
$p((s(0)) \leftarrow : \top$
$p((s(s(X))) \leftarrow p(s(X)) : p(X)$
$q(0) \leftarrow : \forall_X p(X)$

Here the construction of the least constraint model takes precisely $\omega+1$ steps. At step 0 we get $p(0)$ and $p(s(0))$ and at all subsequent steps we get a single new atom. Up to ω we construct all the atoms $p^n(0)$ for $n \in \omega$ and since the Herbrand universe (as opposed to Herbrand base) is ω, the sentence $\forall_X p(X)$ becomes true. Thus the last rule is activated and in the ω^{th} step we get $q(0)$ which completes the construction of the least constraint model.

We shall generalize now our construction of Example 4.1 and for every constructive ordinal α we will produce a program with positive predicate constraints requiring at least α steps to complete construction of the least constraint model. Let $\prec$ be a recursive well-ordering of type α and for each $\beta < alpha$

Intuitively, we want to construct a finite program with positive predicate constraints that at every step β of the construction computes the iβ^{th} element of a recursive well-ordering of type α. This seems to be easy to accomplish by something like

$$p(X) \leftarrow: \forall_Y (Y < X \Rightarrow p(Y))$$

plus a (Horn) program with a binary predicate $<$ computing $\prec$.

Unfortunately, the constraint of our clause is *not* positive. Moreover we need to make sure that when we start to compute $p(\cdot)$ the interpretation of the binary predicate $<$ is correct. In order to overcome the first obstacle we introduce an auxiliary predicate *NotSmaller* with the intended interpretation $N \times N \setminus <$. Then the desired clause takes form

$$p(X) \leftarrow: \forall_Y (\mathit{NotSmaller}(Y, X) \vee p(Y))$$

which clearly is positive. Now we just have to take care of having the correct interpretation of $<$ and *NotSmaller*. Since $<$ is recursive, so is its complement (the intended interpretation of *NotSmaller*) and therefore there are two Horn programs $P_<$ and $P_{\mathit{NotSmaller}}$ computing $<$ and *NotSmaller*, respectively. We can assume that $P_<$ and $P_{\mathit{NotSmaller}}$ are constraint programs (insert $\top$ as constraints). The construction of the least model of the union of $P_< \cup P_{\mathit{NotSmaller}}$ takes ω steps (notice that the construction of Theorem 4.2 collapses to the usual van Emden – Kowalski construction when all the constraints are $\top$). Thus in ω steps we have the correct interpretation of the relations $<$ and *NotSmaller*. We construct now an additional constraint program P' consisting of these two clauses:

$$a \leftarrow: \forall_{X,Y} (Y < X \vee \mathit{NotSmaller}(Y, X))$$
$$p(X) \leftarrow a : \forall_Y (\mathit{NotSmaller}(Y, X) \vee p(Y))$$

and define the program P_α as the union $P_< \cup P_{\mathit{NotSmaller}} \cup P'$.

Proposition 4.4 *The program P_α is a logic program with positive constraints reaching the fixpoint in $\omega + 1 + \alpha$ steps.*

Proof: Clearly in the first ω steps we compute precisely $<$ and *NotSmaller*. a is not computed until the constraint enforcing that $<$ and *NotSmaller* get the correct interpretation is satisfied. Once a is computed (in $(\omega+1)^{st}$ step) we start to compute the consecutive elements of $\prec$. It is clear that in the β^{th} step (after the initial $\omega + 1$ steps) we add to the extent of p the β^{th} element of $\prec$. Since $P_<$ and $P_{NotSmaller}$ do not have in tthe bodies of their clauses neither a nor $p(\cdot)$, the interpretations of $<$ and *NotSmaller* in structures M_ξ for $\xi > \omega$ are always the same. Thus the construction of the least constraint model requires precisely α steps after the initial $\omega + 1$ steps and the thesis is proven. □

Using the usual logic programming paradigm it is natural to look at the operator associated with the program.

Here is the definition of the operator. The variable I ranges over the subsets of the Herbrand base. $\models$ denotes the satisfaction relation between the formulas used for constraints and subsets of Herbrand base.
$S_P(I) = \{p : \text{for some } q_1, \ldots, q_n \in I \text{ and } I \models \Phi,\ p \leftarrow q_1, \ldots, q_n : \Phi \text{ belongs to } ground(P)\}$

Proposition 4.5 *If all the constraints in P are preserved upwards, then S_P is a monotone operator. Hence S_P possesses the least and the largest fixpoints.*

It is not difficult to see that the least fixpoint of S_P is constructed in the proof of Theorem 4.2. Thus this least fixpoint is the least constraint model of P.

The largest fixpoint of P is also a constraint model but of a transformed program. This is the case because we did not introduce the notion of a supported model and so we need to make a program transformation first.

Specifically, given a clause with a logical constraint

$$C = p \leftarrow q_1, \ldots, q_n : \Phi$$

let C^{IV} be the clause

$$C^{IV} = p \leftarrow : q_1 \wedge \ldots \wedge q_n \wedge \Phi$$

We then have the following fact:

Proposition 4.6 *Assume that all the constraints in clauses of P are preserved upwards. Then fixpoints of operator S_P are precisely constraint models of P^{IV}. Thus the largest fixpoint of S_P is the largest constraint model of P^{IV}.*

We noticed that the least fixpoint of S_P is the least constraint model for P provided that all the constraints of P are preserved upwards. Yet, the operator S_P does not need to be compact and so the fixpoint may be reached in more than ω steps. The influence of constraints on the actual closure ordinal of both downward and upward iteration (see Blair [6] for the results on the closure ordinal of downward iteration) is not clear at this point.

Now let us look at constraints preserved downwards. Formally, Φ is preserved downwards if $M_1 \subseteq M_2$ and $M_2 \models \Phi$ implies $M_1 \models \Phi$. Clearly, purely negative formulas of propositional language are preserved downwards but in other languages there are many more such formulas. For instance any negation of a purely positive formula is preserved downwards.

We have the following proposition.

Proposition 4.7 *If all the constraints in the constraint program P are preserving downwards then the constraint models of P are inclusion-incompatible.*

Proof: Let $M_1 \subseteq M_2$ be two constraint models of P. Since all the constraints are preserving downwards therefore $P_{M_2} \subseteq P_{M_1}$. Therefore $N_{M_2} \subseteq N_{M_1}$ and since $M_1 = N_{M_1}$ and $M_2 = M_{M_2}$, $M_2 \subseteq M_1$. Thus $M_1 = M_2$. □

5 Proof theory for constraint programs

Although Logic Programming is often associated with top-down evaluation of queries (i.e. backward chaining), we can look at the usual Horn programming

in an forward chaining fashion (i.e. with bottom-up evaluation). In this paradigm we treat the clauses of the program as rules of proof which fire when their premises are proven. This is how the layers of the iterations of the operator T_P are produced.

In case of programs with logical constraint a similar construction is possible except that we need to take into account the fact that the application of rules requires that its constraint be satisfied.

We will assume now that all the constraint come from some formal language. The nature of this language is immaterial as long as we have a satisfaction relation between the subsets of the Herbrand base and formulas of that language.

We define the notion of M-proof of an atom p from program P (here M is a subset of Herbrand base) inductively.

Definition 5.1 1. If $p \leftarrow: \Phi$ belongs to P and $M \models \Phi$ then $\langle p \rangle$ is an M-proof of length 1.

2. If $p \leftarrow q_1, \ldots, q_n : \Phi$ belongs to P and $S_1, \ldots, S_n$ are M-proofs, respectively, of $q_1, \ldots, q_n$ of length, respectively, $m_1, \ldots, m_n$ and $M \models \Phi$ then $S_1 \frown \ldots \frown S_n \frown \langle p \rangle$ is an M-proof of p of length $m_1 + \ldots + m_n + 1$.

3. p is M-provable from P if p has an M-proof of some length from P.

With this concept we have the following characterization theorem

Theorem 5.2 *M is a constraint model of P if and only if M coincides with the set of atoms possessing an M-proof from P.*

Proof: We prove two facts:
(a) All elements with M-proof from P belong to N_M.
This is easily proven by induction on the length of such M-proof.
(b) All elements of N_M possess an M-proof from P.
This is again proven by induction, but this time on the least number n such that p belongs to $T_{P_M} \uparrow n(\emptyset)$.
It follows that M coincides with the least model of P_M if and only if M consists of all elements possessing an M-proof from P. □

6 Default Logic and programs with logical constraints

Our approach to logical constraints allows us to get an insight into Reiter's Default Logic [20] and on some possible extensions of Reiter's formalism.

Recall that a default rules is a rule of the form:

$$d = \frac{\alpha : M\beta_1, \ldots, M\beta_k}{\gamma}$$

A default theory is a pair $\langle D, W\rangle$ where D is a set of default rules and W is a set of formulas.

We shall see now how logic programs with logical constraints can be used to get a nice rendering of default logic. Our Herbrand base (the set of atoms) is the set of all formulas of the propositional language $\mathcal{L}$. The constraints, however, are formulas of the form

$$\neg\delta_1 \notin S \& \ldots \& \neg\delta_k \notin S \tag{1}$$

We need to define when a set of atoms, say I, satisfies a constraint of the form (1). It is defined in the most natural way; namely $I \models \neg\delta_1 \notin S \& \ldots \& \neg\delta_k \notin S$ if none of $\neg\delta_i$, $1 \leq i \leq k$ belongs to I,

Now we translate the rule d into the following clause with logical constraints:

$$t(d) = \gamma \leftarrow \alpha : \neg\beta_1 \notin S \& \ldots \& \neg\beta_k \notin S$$

Two more sets of rules must be included in addition to the translations of the default rules in D. First we need to translate the formulas of W. This is done by adding clauses of the form

$$t(\gamma) = \gamma \leftarrow : \top$$

for each $\gamma \in W$. Moreover we need to add the processing rules of logic. This is done by a (uniform) construction of an auxiliary program P_{logic}.

$\alpha \leftarrow : \top$ for all tautologies α, and
$\beta \leftarrow \alpha \supset \beta, \alpha \ : \top$ for all pairs of formulas α and β.

Now form a translation of the default theory $\langle D, W\rangle$ as follows:

$$t(D,W) = \{t(d) : d \in D\} \cup \{t(\gamma) : \gamma \in W\} \cup P_{\text{logic}}$$

We then have the following result.

Theorem 6.1 *Let $\langle D, W\rangle$ be a default theory, and let $t(D,W)$ be its translation as a constraint program as defined above. Then I is a constraint model of $t(D,W)$ if and only if I is a default extension of $\langle D, W\rangle$.*

We notice that once this result is proved, it becomes clear that Reiter's Default Logic is just one of many other techniques for constructions of constraint-controlled rules of proof in propositional logic. We shall not pursue the matter further here, although it should be clear that the results on upward preserved constraints are also applicable for theories with such constraints.

7 Constraint Models for Programs with Downward-Preserving Constraints

We shall outline now a construction of a constraint model for the case of programs with downward-preserving constraints. These programs, in contrast to programs with upward-preserving constraints do not necessarily possess constraint models.

We shall show now a construction which is not necessarily sound but it is complete in important cases. This construction is a modification of forward chaining construction of stable model of logic program.

In our case the construction may require well-orderings of the length longer than ω. Therefore we will have to assure preservation of constraints used in the construction under limits of increasing sequences. This requires that we introduce the following definition.

Definition 7.1 *We say a formula Φ is limit-preserving if the following holds: Whenever $\langle M_\xi\rangle_{\xi<\alpha}$ is increasing family of subsets of Herbrand base (i.e for $\xi_1 < \xi_2\alpha$, $M_{\xi_1} \subseteq M_{\xi_2}$) and for all $\xi < \alpha$, $M_\xi \models \Phi$, then $\bigcup_{\xi<\alpha} M_\xi \models \Phi$.*

First-order formulas limit-preserving formulas are Π_2 formulas, that is formulas which in their prenex form have one at most one alternation of quantifiers with universal quantifier as the first one.

Limit-preserving formulas are closed under conjunctions and alternatives (finite or infinitary). It should be clear that atomic sentences are limit preserved.

We shall describe now a construction which given a constraint program will always produce a constraint model of a subprogram of the original constraint program even in the case when the original program has no constraint models. For constraints which are (disjunctions, even possibly infinite) of negative literals, we prove a completeness result as well. Specifically, we will show that in that case every constraint model can be obtained by means of our construction.

Let P be a constraint program. Let $\preceq$ be a well-ordering of ground version of P. We also assume that all the constraints in clauses of P are limit-preserving. By $H(P)$ we mean the part of program P consisting of these clauses C for which Φ_C is a tautology. That is $H(P)$ is the Horn part of P. Thus $T_{H(P)}$ is an operator associated with the Horn part of P [21]. The least fixpoint of this operator is the least model of $H(P)$. We define an increasing family of subsets of Herbrand base of P, $\langle A_\xi \rangle$ and a sequence of clauses $\langle C_\xi \rangle$ as follows:

Definition 7.2 $A_0 = T_{H(P)} \uparrow \omega(\emptyset)$.

If α is limit ordinal and A_ξ are defined for ordinals $\xi < \alpha$, then $M_\alpha = \bigcup_{\xi<\alpha} M_\xi$.

If α is a non-limit ordinal, say $\alpha = \beta + 1$, and A_β already defined, then consider clauses C with the following properties:

1. *head*$(C) \notin A_\beta$,
2. *body*$(C) \subseteq A_\beta$
3. $A_\beta \models \Phi_C$

4. $T_{H(P)} \uparrow \omega(A_\beta \cup \{head(C)\}) \models \Phi_\xi$ *for all* $\xi < \alpha$

5. $T_{H(P)} \uparrow \omega(A_\beta \cup \{head(C)\}) \models \Phi_C$

If there is no clause C satisfying all the above conditions then C_α is not defined, whereas $A_\alpha = \bigcup_{\xi<\alpha} A_\xi$.

Otherwise C_α is the $\prec$-first clause satisfying the above conditions and we also define

$$A_\alpha = T_{H(P)} \uparrow \omega(A_\beta \cup \{head(C)\})$$

The set $A = \bigcup A_\alpha$ is called the set of accepted elements. The least ordinal α such that C_α is not defined is called the length of the construction. This ordinal number α depends on the ordering $\prec$ and we denote it $\alpha_\prec$.

It is easy to see that $A = \bigcup_{\xi<\alpha_\prec} A_\xi$.

We say that a clause C is called contradictory if $head(C) \notin A$, $body(C) \subseteq A$, $A \models \Phi_C$ but $T_{H(P)} \uparrow \omega(A \cup \{head(C)\})$ does not satisfy Φ_{C_ξ} for some $\xi < \alpha_\prec$ or $T_{H(P)} \uparrow \omega(A \cup \{head(C)\})$ does not satisfy Φ_C.

The contradictory clauses are clauses which met all the preconditions for application, but after it has been fired either some of the constraints were previously applied are no longer valid or the constraint of C (which was valid in A) is no longer valid.

Whether a clause is contradictory depends on $\prec$. A clause can be non-contradictory for various reasons. For instance its head may already be in A. Then there is no need to look at the body. Or the body of C may be not satisfied in A or, finally, the constraint of C may be false in A. Finally, a clause which has been applied during the construction of A is non-contradictory.

Let us denote by $I^\prec$ the set of clauses that are contradictory.

Theorem 7.3 *1. Let $\prec$ be a well-ordering of P. Then $A^\prec$ is a constraint model of $P \setminus I^\prec$.*

2. If all the constraints in P are conjunctions of negative literals (finite or infinite) or disjunctions of conjunctions of negative literals, then for

every constraint model M of P there exists a well-ordering $\prec$ of P such that $A^{\prec} = M$.

The operator S_P introduced in Section 4 for positive programs can be studied for arbitrary programs. For downward-preserving constraints the operator S_P is no longer monotone. In fact, it is antimonotone.

Proposition 7.4 *When all the constraints in P are downward-preserving, S_P is antimonotone operator.*

Therefore, as noticed in [5], the operator S_P^2 is monotone. Thus S_P^2 possesses both least and largest fixpoints. Moreover, these fixpoints, say A and B, are alternating, i.e. $S_P(A) = B$ and $S_P(B) = A$. Thus, although we cannot be sure that that there is a constraint model (as was the case of programs with upward-preserving constraints), we have an alternating pair. Notice that $S_P(A)$ consists of these atoms that can be computed when the constraints are checked with respect to A.

8 Constraint disjunctive programming

In this section we generalize programs with logical constraints to accomodate "nonstandard" disjunction in the heads of program clauses. That is we introduce an additional construct proposed by Gelfond and Lifschitz [14], see also Baral and Gelfond [4]. Surprisingly, it turns out that if the nonstandard disjunction is permitted and all the constraints are propositional we obtain precisely the class of "general disjunctive programs" of Lifschitz and Woo [17].

Let us define a disjunctive clause with logical constraint as a construct of the form

$$p_1 | \ldots | p_k \leftarrow q_1, \ldots, q_m : \Phi$$

where $p_1, \ldots, p_k, q_1, \ldots, q_m$ are atoms and Φ is a formula of some language for which a satisfaction relation $\models$ is defined. This satisfaction relation $\models$ maps the pairs consisting of subsets of the set of atoms and formulas into $\{0, 1\}$.

A disjunctive constraint program is a set of disjunctive clauses with logical constraint.

We shall now define the notion of constraint model of a disjunctive constraint program. To this end we first recall the notion of an answer set for a disjunctive program as defined in [14]. Specifically, a subset $M \subset At$ is called closed under a disjunctive rule

$$p_1| \ldots |p_k \leftarrow q_1, \ldots, q_m$$

if whenever all q_i, $1 \leq i \leq m$ belong to M then at least one of p_j, $1 \leq j \leq k$ belongs to M as well. An answer set of a set P of disjunctive rules is a minimal set M closed under all the rules in P. With this definition we define the notion of a constraint model of disjunctive constraint program P as follows. Let $M \subseteq At$. Consider P^M consisting of disjunctive rules $p_1| \ldots |p_k \leftarrow q_1, \ldots, q_m$ such that for some Φ, $M \models \Phi$ and $p_1| \ldots |p_k \leftarrow q_1, \ldots, q_m : \Phi$. If M is an answer set for P^M then M is called a constraint model for P.

We notice that Propositions 3.2 and refp.3 hold in the present context. That is substitution of equivalent constraints and splitting disjunctions in the context part does not change the semantics.

Let us now consider the case when all the constraints are propositional formulas. Then, given a disjunctive constraint program P, there is a disjunctive program P' such that the collections of constraint models of P and P' coincide and all the constraints of P' are of the form $l_1 \wedge \ldots \wedge l_s$ where $l_1, \ldots, l_s$ are literals.

Thus, up to a syntactic transformation discussed above, we are dealing with the programs consisting of program clauses

$$p_1| \ldots |p_k \leftarrow q_1, \ldots, q_m : t_1, \wedge \ldots \wedge t_l \wedge \neg r_1 \wedge \ldots \wedge \ldots r_n$$

We shall now see that these programs coincide, up to syntactic sugar, with general disjunctive programs of [17]. That is we will show a simple modular translation of general disjunctive programs to constraint disjunctive programs so that the answer sets are precisely constraint models of the translated programs.

To this end recall that a general disjunctive rule is a rule of the form:

$$C = p_1| \ldots |p_k|\mathbf{not}(t_1)| \ldots !\mathbf{not}(t_l) \leftarrow q_1, \ldots, q_m, \mathbf{not}(r_1), \ldots, \mathbf{not}(r_n)$$

A general disjunctive program is a set of such clauses. An answer set to such program is constructed as follows: Given $M \subset At$, P^M consists of clauses

$$p_1 | \ldots | p_k \leftarrow q_1, \ldots, q_m$$

such that for some $t_1, \ldots, t_l$ in M and for some $r_1, \ldots, r_n$ all not in M,

$$p_1 | \ldots | p_k | \mathbf{not}(t_1) | \ldots ! \mathbf{not}(t_l) \leftarrow q_1, \ldots, q_m, \mathbf{not}(r_1), \ldots, \mathbf{not}(r_n)$$

belongs to P. M is an answer set for P if M is an answer set for P^M.

We define now the translation as follows. A general disjunctive clause

$$C = p_1 | \ldots | p_k | \mathbf{not}(t_1) | \ldots ! \mathbf{not}(t_l) \leftarrow q_1, \ldots, q_m, \mathbf{not}(r_1), \ldots, \mathbf{not}(r_n)$$

is translated to a disjunctive constraint clause:

$$C^v = p_1 | \ldots | p_k \leftarrow q_1, \ldots, q_m : t_1 \wedge \ldots \wedge t_l \wedge \neg r_1 \wedge \ldots \wedge \wedge r_n$$

We define $P^v = \{C^v : C \in P\}$. We have then the following result

Proposition 8.1 *Let P be a general disjunctive program and P^v its translation as a constraint disjunctive program. Then for every $M \subseteq At$, M is an answer set for P if and only if M is a constraint model for P^v.*

Proof: Given $M \subseteq At$, it is easy to check that

$$C = p_1 | \ldots | p_k \leftarrow q_1, \ldots, q_m$$

belongs to P^M if and only if if

$$C' = p_1 | \ldots | p_k \leftarrow q_1, \ldots, q_m : \top$$

belongs to $(P^v)^M$. Since in both cases we look for minimal sets closed under rules (and all constraints in $(P^v)^M$ are trivial), we get the desired result. □

Proposition 8.1 allows us to use the results of Eiter and Gottlob [12] to determine the complexity of various problems related constraint models of constraint disjunctive programs.

By existence problem for semantics $\mathcal{S}$ (and a class of programs $\mathcal{C}$) we mean the problem of finding if the program $P \in \mathcal{C}$ possesses a model in the class $\mathcal{S}$. Similarly, the entailment problem for $\mathcal{S}$ is checking whether (given a program $P \in \mathcal{C}$ and a formula φ) φ is true in models of P from $\mathcal{S}$. For the semantics of answer sets and the class DP, of disjuntive programs without $/$ in the heads of clauses, Eiter and Gottlob [12] determined the complexity of both existence and entailment problems.

Proposition 8.2 (Eiter and Gottlob [12]) *Existence problem for answer sets of disjunctive programs is Σ_2^P-complete. The entailment problem for these classes is Π_2^P-complete.*

We will use this result together with our translation result (Proposition 8.1) to establish the results on complexity of the existence and entailment problems for constraint disjunctive programs.

Proposition 8.3 *The existence of constraint models of constraint disjunctive programs is a Σ_2^P-complete problem. The corresponding entailment problem is Π_1^P-complete.*

Proof: We discuss the existence problem, the entailment problem is similar. First, notice that our problem belongs to the class Σ_2^P. Indeed, after the initial guess of M, the reduction process takes linear time. Next, we need to check if M is a minimal model of P^M which can be done with one call to NP-oracle. Second, we establish Σ_2^P-hardness of our problem. Indeed, the existence problem for answer sets for disjunctive programs reduces to our problem by Propositions 8.1 and 8.2. This estabilshes that our problem is Σ_2^P-complete. □

9 Predicate programs over a recursive, decidable, domain

In this section we will look at predicate programs where the constraint checking is done with respect to some recursive and decidable domain. We will

see that the least constraint model assigned to such program (see below for details) has all relations recursively enumerable.

We first need to develop a theory corresponding to Herbrand structures, but without the restrictions that different ground terms denote different elements. Thus, in particular, Clark's equality axioms will not hold in our Herbrand models.

Let $\mathcal{A} = \langle A, R_1, \ldots, R_k, f_1, \ldots, f_l, a_1, \ldots \rangle$ be a relational structure and $\langle A, \ldots, f_l, a_1, \ldots \rangle$ be its underlying algebra. We say that this algebra (and by abuse of language $\mathcal{A}$) is *generated by constants* if every element $b \in A$ is the value of some ground term $t(a_{i_1}, \ldots, a_{i_n})$.

Now, we define the Herbrand universe U and Herbrand base H of the language $\mathcal{L}_\mathcal{A}$ in the usual manner. We define in this universe a structure $\mathcal{H}_\mathcal{A}$ as follows:

$$R'_i(t_1, \ldots, t_m) \text{ iff } t_1^\mathcal{A} = b_1, \wedge \ldots \wedge t_m^\mathcal{A} = b_m \wedge R_i(b_1, \ldots, b_m)$$

$$Eq(t_1, t_2) \text{ iff } t_1^\mathcal{A} = t_2^\mathcal{A}$$

Let $\mathcal{H}_\mathcal{A} = \langle U, R'_1, \ldots R'_k, f_1, \ldots f_l, a_1, \ldots \rangle$ be a structure on Herbrand universe with Herbrand interpretation of functions and relations defined above. We then have:

Proposition 9.1 *Let $\mathcal{A}$ be a structure generated by constants. and let $\mathcal{H}_\mathcal{A}$ be the structure over Herbrand universe defined above. Then*

1. *$\mathcal{H}_\mathcal{A}$ is elementarily equivalent to $\mathcal{A}$*
2. *The mapping contract : $t \mapsto t^\mathcal{A}$ is a homomorphism of $\mathcal{H}_\mathcal{A}$ onto $\mathcal{A}$.*

Proof: The following stronger fact implies both parts. If φ is a formula with at most k free variables then:

$$\mathcal{H}_\mathcal{A} \models \varphi[t_1, \ldots, t_k] \text{ iff } \mathcal{A} \models \varphi[t_1^\mathcal{A}, \ldots t_k^\mathcal{A}]$$

This fact is easily proved by induction. The fact that the structure is generated by constants is used in the quantifier case. □

Usually, the subsets of Herbrand base correspond to relational structures. In our case, we need to fine tune this correspondence. Namely, we consider subsets of H closed under Eq. These are subsets M of $H_{\mathcal{A}}$ satisfying this property:

$$r_i(t_1, \ldots, t_k) \in M \wedge Eq(t_1, s_1) \wedge \ldots \wedge Eq(t_k, s_k) \Rightarrow r_i(s_1, \ldots, s_k) \in M$$

There is a one-to-one correspondence between closed subsets of H and relational structures on the universe of the underlying algebra. We will use this fact below.

Now, let $\mathcal{A} = \langle A, R_1, \ldots, R_k, f_1, \ldots, f_l, a_1, \ldots \rangle$ be a relational structure. We say that $\mathcal{A}$ is decidable recursive if:

1. $A \subseteq N$ is recursive
2. All R_i, f_j are recursive relations and functions
3. $\langle a_1, \ldots, \rangle$ is recursive
4. $\mathcal{A}$ is decidable, that is there is an algorithm that, given any integer k, a formula φ with free variables among $v_0, \ldots, v_{k-1}$ and a sequence s of elements of A of length k, correctly decides if $\mathcal{A} \models \varphi[s]$
5. $\mathcal{A}$ is generated by constants.

Now, let P be a predicate constraint program over a domain $\mathcal{A}$. A grounding of P, *ground*(P) is the set of all ground substitutions of all clauses of P incremented by the following auxiliary program $P_{\mathcal{A}}$. Namely $P_{\mathcal{A}}$ consists of clauses of three types:
(i) $Eq(t_1, t_2) \leftarrow : \top$, whenever $t_1^{\mathcal{A}} = t_2^{\mathcal{A}}$
(ii) $r_i(t_1, \ldots, t_n) \leftarrow : \top$ whenever $\langle t_1^{\mathcal{A}}, \ldots, t_n^{\mathcal{A}} \rangle \in R_i$
(iii) $s_i(t_1, \ldots, t_n) \leftarrow s_i(t'_1, \ldots, t'_n), Eq(t_1, t'_1), \ldots, Eq(t_n, t'_n)$ for all s_i's and all choices of ground terms.
The goal of clauses of types (i) and (ii) is to reconstruct all diagram of $\mathcal{A}$. The clauses of type (iii) make sure that the subset of extended Herbrand base that will be computed is closed.

Now we say that a predicate constraint program P over domain $\mathcal{A}$ is *simple* if:

1. The predicate letters $r_1, \ldots, r_k$ (that is names of relations in $\mathcal{A}$) do not appear in heads of clauses from P

2. The only function symbols appearing in P are those of $f_1, \ldots, f_l$ that is the only function symbols of $\mathcal{L}_{\mathcal{A}}$ are allowed.

Thus simple programs do not attempt to redefine relations from $\mathcal{A}$. This is, of course, entirely analogous to DATALOG situation. Relations definable in $\mathcal{A}$ are treated as extensional relations.

Let $Diag^{\mathcal{A}} = \bigcup_i \{r_i(t_1, \ldots, t_n) : \langle t_1^{\mathcal{A}}, \ldots, t_n^{\mathcal{A}} \rangle \in R_i\}$.

The following lemma says that constraints simple programs depend on $\mathcal{A}$ only.

Lemma 9.2 *Let P be a simple program and let M_1, M_2 be two subsets of extended Herbrand base such that $M_1 \cap Diag^{\mathcal{A}} = M_2 \cap Diag^{\mathcal{A}}$. Then $P^{M_1} = P^{M_2}$.*

Lemma 9.2 says that with simple programs the part of the set used for reduction which is outside of $Diag^{\mathcal{A}}$ does not matter.

We now have the result showing that with simple programs the least model we compute has only recursively enumerable relations.

Proposition 9.3 *If $\mathcal{A}$ is a decidable recursive structure and P a simple program, then all relations of the least constraint model of P are recursively enumerable.*

Proof: First of all, notice that P possesses a least constraint model. It reconstructs all relations R_i (notice the way we defined the grounding of the program). Since these relations are not redefined, the least constraint model $\mathcal{B}$ has all relations R_i $(1 \leq i \leq k)$ identical with those of $\mathcal{A}$. Since $\mathcal{A}$ is recursive, these relations are recursive, thus recursively enumerable.

We show now that the relations S_j (interpreting predicates occurring in the heads of clauses from P) are also recursively enumerable.

To this end, tle $\langle b_1, \ldots, b_n \rangle$ be a tuple of elements from A (notice that the universe of $\mathcal{B}$ is A). Since $\mathcal{A}$ is generated by constants, there is a tuple $\langle t_1, \ldots, t_n \rangle$ so that $t_i^{\mathcal{A}} = b_i$ for $1 \leq i \leq n$. Take such tuple of terms $\langle t_1, \ldots, t_n \rangle$. Now notice that the program $P^{Diag(\mathcal{A})}$ is recursively enumerable. The reason is that $Diag(\mathcal{A})$ is recursive.

Thus there is a recursive function L listing all the atoms of the form $s_i(t_1, \ldots, t_n)$ which are computed in the computation of all terms in the least model of $P^{Diag^{\mathcal{A}}}$. This is so, because all iterations of the T-operator for $P^{Diag^{\mathcal{A}}}$ are recursively enumerable, and in uniform way.
Now we run L on consecutive inputs. If we compute $s_i(t_1, \ldots, t_n)$, we put answer the query $\langle b_1, \ldots, b_n \rangle \in S_i$ in positive. Thus S_i is recursively enumerable. □

Notice that this result is optimal in the sense that all the recursively enumerable sets can be computed with Horn programs and with some small work the usual argumment can be lifted to the present context.

References

[1] A.V. Aho and J.D. Ullman. Universality of Data Retrieval Languages. In: ACM Symposium on Principles of Programming Languages 2979, pages 110–120.

[2] K. Apt. Logic programming. In: J. van Leeuven, editor, *Handbook of Theoretical Computer Science*, pages 493–574. MIT Press, Cambridge, MA, 1990.

[3] K.R. Apt, H.A. Blair, and A. Walker. Towards a Theory of Declarative Knowledge. In: J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–142, Los Altos, CA, 1987. Morgan Kaufmann.

[4] C. Baral and M. Gelfond, Logic programming and knowledge representation. *Journal of Logic Programming* 19-20 pages 73-148, 1994.

[5] C. Baral and V.S. Subrahmanian. Stable and Extension Class Theory for Logic Programs and Default Theories. *Journal of Automated Reasoning* 8, pages 345–366, 1992.

[6] H.A. Blair. The recursion-theoretic complexity of predicate logic as programming language. *Information and Control* 54, pages 25–47, 1982.

[7] A.K. Chandra and D. Harel. Structure and complexity of relational queries. *Journal of Computer and System Sciences* 43, pages 99–128, 1982.

[8] C.C. Chang and H.J. Keisler. *Model Theory.* North Holland, Amsterdam, 1972.

[9] K.L. Clark. Negation as failure. In: H. Gallaire and J. Minker, editors, *Logic and data bases*, pages 293–322. Plenum Press, 1978.

[10] A. Colmerauer. PROLOG III Reference and Users Manual, PrologIA, Marseilles 1990.

[11] M. Dincbas and H. Simonis and P. Van Hententryck and A. Aggoun. The Constraint Logic Programming Language CHIP. In: *Proceedings of the 2nd International Conference on Fifth Generation Computing Systems*, pages 249-264. 1988.

[12] T. Eiter and G. Gottlob. On the Computational Cost of Disjunctive Logic Programming: Propositional Case. *Annals of Mathematics and Artificial Intelligence*, to appear.

[13] M. Gelfond and V. Lifschitz. The Stable Semantics for Logic Programs. In: *Proceedings of the 5th International Symposium on Logic Programming*, pages 1070–1080, Cambridge, MA., 1988. MIT Press.

[14] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9, pages 365–385, 1991.

[15] J. Jaffar and J.-L. Lassez. Constraint Logic Programming. In: *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*, pages 111–119, Münich, 1987.

[16] J. Jaffar and M. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming* 19-20, pages 503–581, 1994.

[17] V. Lifschitz and T.Y.C. Woo. Answer Sets in General Nonmonotonic Reasoning. *Proceedings of the 3rd international conference on principles of knowledge representation and reasoning, KR '92*, pages 603–614, San Mateo, CA., 1992, Morgan Kaufmann.

[18] A.R. Meyer and L.J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential time, *Proceedings of the 13th annual symposium on switching and automata theory*, pages 125–129, IEEE Computer Society, 1972.

[19] C. Pollett and J.B. Remmel. Logic programs with quantified boolean constraints. Unpublished manuscript, 1995.

[20] R. Reiter. A logic for default reasoning. *Artificial Intelligence* 13, pages 81–132, 1980.

[21] M.H. van Emden and R.A. Kowalski. The Semantics of Predicate Logic as a Programming Language. *Journal of the ACM* 23 pages 733–742, 1976.