$\mathbb{FDNC}$: Decidable Non-monotonic Disjunctive Logic Programs with Function Symbols[⋆]

Mantas Šimkus and Thomas Eiter

Institut für Informationssysteme, Technische Universität Wien
Favoritenstraße 9-11, A-1040 Vienna, Austria
(simkus,eiter)@kr.tuwien.ac.at

Abstract. Current Answer Set Programming systems are built on non-monotonic logic programs without function symbols; as well-known, they lead to high undecidability in general. However, function symbols are highly desirable for various applications, which challenges to find meaningful and decidable fragments of this setting. We present the class $\mathbb{FDNC}$ of logic programs which allows for function symbols, disjunction, non-monotonic negation under answer set semantics, and constraints, while still retaining the decidability of the standard reasoning tasks. Thanks to these features, they are a powerful formalism for rule-based modeling of applications with potentially infinite processes and objects, which allows also for common-sense reasoning. We show that consistency checking and brave reasoning are ExpTime-complete in general, but have lower complexity for restricted fragments, and outline worst-case optimal reasoning procedures for these tasks. Furthermore, we present a finite representation of the possibly infinitely many infinite stable models of an $\mathbb{FDNC}$ program, which may be exploited for knowledge compilation purposes.

1 Introduction

Answer Set Programming (ASP) is a declarative programming paradigm which has its roots in Logic Programming and Non-monotonic Reasoning. It is well-suited for modeling and solving problems which involve common sense reasoning, and has been fruitfully applied to a range of applications including data integration, configuration, reasoning about actions and change, etc.; see [16].

While Answer Set Semantics, which underlies ASP, was defined in the setting of a general first-order language, current ASP frameworks and implementations, like DLV [10], Smodels [15], and other efficient solvers are based on function-free languages and resort to Datalog with negation and its extensions. However, it is widely acknowledged that this leads to drawbacks related to expressiveness, and also to inconvenience in knowledge representation, cf. [2]. Since one is forced to work with finite domains, potentially infinite processes cannot be represented naturally in ASP. Additional tools must be used, which may incur high space requirements.

⋆ Work partially supported by the Austrian Science Funds (FWF) project P17212 and the EC project REWERSE (IST-2003-506779).

Function symbols, in turn, are a very convenient means for generating infinite domains and objects, and allow for more natural representation of problems on such domains. However, they have been banned in ASP for a good reason, since they quickly lead to undecidability even for Horn programs, and with negation under the answer set semantics, they lead to high undecidability, cf. [11,12]. This raises the challenge to single out meaningful fragments of ASP with function symbols which allow to model infinite domains while still retaining the decidability of the standard reasoning tasks. Important work on this is by Bonatti and his colleagues on their *finitary programs* and *finitely recursive programs*[2,1], which impose syntactic conditions on the groundings of logic programs. However, the hardness of verifying the satisfaction of the conditions limits the applicability of the results; see Section 5 for more discussion.

In this paper, we pursue an approach to obtain decidable logic programs with function symbols by merely constraining the syntax in a way that can be effectively checked. To this end, we take inspiration from results in automated deduction and other areas of knowledge representation, where many procedures, like tableaux algorithms with blocking, or hyper-resolution, have been developed for deciding satisfiability in various fragments of first-order logic. When function symbols (or existential quantification) may occur, these procedures are often sophisticated since they must deal with possibly infinite models. However, because of the peculiarities of Answer Set Semantics, transferring these results to logic programs is not straightforward. Reasoning with logic programs needs to be more refined since only minimal (or stable) models count as models of a given program. Our main contributions are briefly summarized as follows.

- We introduce the class 𝔽𝔻ℕℂ of logic programs, which allow for function symbols, disjunction, constraints, and non-monotonic negation under the answer set semantics [6]. The restrictions we apply are syntactic and ensure that programs have a *forest-shaped model* property. 𝔽𝔻ℕℂ programs are a convenient tool for knowledge representation. They allow, e.g., the representation of an evolving action domain (see Section 3).
- We show that standard reasoning tasks are decidable for 𝔽𝔻ℕℂ, and are ExpTime-complete; this includes checking the consistency (i.e., the existence of a stable model), and brave entailment of ground atomic or existential atomic queries. Disallowing disjunction and constraints (𝔽ℕ) or non-monotonic negation (𝔽𝔻ℂ) does not lead to lower complexity, i.e., the problems considered remain ExpTime-complete. Depending on the reasoning task, reasoning is at most PSpace-complete for further restricted classes.
- Noticeably, the hardness proofs for consistency checking in 𝔽ℕ, 𝔽𝔻ℂ, and 𝔽𝔻ℕℂ, are by a reduction from satisfiability testing in the ExpTime-complete Description Logic $\mathcal{ALC}$. Thus, as a side result we obtain a novel polynomial time mapping of a well-known Description Logic to logic programming.
- 𝔽𝔻ℕℂ programs can have infinitely many and infinitely large stable models, which therefore can not be explicitly represented. We provide a method to finitely represent all the stable models of a given 𝔽𝔻ℕℂ program. This is

achieved by a composition technique that allows to reconstruct stable models as forests, i.e., sets of trees, from *knots*, which are instances of generic labeled trees of depth 1. The finite representation technique allows us to define an elegant decision procedure for brave reasoning in $\mathbb{FDNC}$, and may also be exploited for offline knowledge compilation to speed up online reasoning, by precomputing and storing the knots of a program.

Thanks to their features, $\mathbb{FDNC}$ programs are a powerful formalism for rule-based modeling of applications with a potentially infinite domain, which also accommodates common-sense reasoning through non-monotonic negation. From a complexity perspective, $\mathbb{FDNC}$ and its subclasses offer *effective syntax* for encoding problems in PSpace and ExpTime to logic programs with function symbols.

2 Preliminaries

A *disjunctive rule* (briefly, *rule*) is an expression of the form

$$A_1 \vee \ldots \vee A_n \leftarrow L_1, \ldots, L_m,$$

where $n{+}m > 0$, $A_1, \ldots, A_n$ are *atoms* and $L_1, \ldots, L_m$ are *literals.* The atoms are from a standard first-order language with countably infinite sets of *variables*, *constant symbols*, and *function* and *predicate symbols* of positive arity. A literal is either an atom A (*positive literal*), or an expression *not* A (*negative literal*). The atoms $A_1, \ldots, A_n$ are *head atoms*, while $L_1, \ldots, L_m$ are *body literals.* For a rule r, let $\mathsf{head}(r)$, $\mathsf{body}^+(r)$, and $\mathsf{body}^-(r)$ respectively denote the set of its head atoms, positive body literals, and negative body literals. We say r is a *fact*, if $n = 0$; a *constraint*, if $m = 0$; and *positive*, if $\mathsf{body}^-(r) = \emptyset$.

A *disjunctive logic program* (briefly, *program*) is an arbitrary set of rules. It is *positive* (resp., *ground*), if contains only positive (resp., ground) rules. For a program P, its *Herbrand universe*, *Herbrand base* and its ground instantiation are defined in the standard way, and are respectively denoted by $\mathcal{HU}^P$, $\mathcal{HB}^P$ and $\mathsf{Ground}(P)$; see [13]. Furthermore, by $MM(P)$ we denote the set of *(Herbrand) interpretations* that are (set-inclusion) minimal models of a ground positive program P.

An interpretation I of a program P is a *stable model* of P iff $I \in MM(P^I)$, where P^I is the *Gelfond-Lifschitz reduct* [6] of P, obtained from $\mathsf{Ground}(P)$ by removing (i) each rule r such that $\mathsf{body}^-(r) \cap I \neq \emptyset$, and (ii) all the negative literals from the remaining rules. The set of stable models of a program P is denoted by $SM(P)$.

A program P is *consistent*, if $SM(P) \neq \emptyset$. A program P *bravely entails* a ground (variable-free) atom A (in symbols, $P \models_b A$), if some stable model I of P contains A. An *existential atomic query* is an expression $\exists \boldsymbol{x}.A(\boldsymbol{x})$, where $\boldsymbol{x}$ is a n-tuple of variables and A is a predicate symbol of arity n. A program P *bravely entails* $\exists \boldsymbol{x}.A(\boldsymbol{x})$ (in symbols, $P \models_b \exists \boldsymbol{x}.A(\boldsymbol{x})$), if some stable model I of P contains a ground atom $A(\boldsymbol{t})$.

(1) $Change(x, grow(x)) \leftarrow Young(x), Warm(x)$
(2) $Change(x, cell_1(x)) \leftarrow Mature(x), Warm(x)$
(3) $Change(x, cell_2(x)) \leftarrow Mature(x), Warm(x)$
(4) $Change(x, die(x)) \leftarrow Cold(x)$
(5) $Young(cell_1(x)) \leftarrow Change(x, cell_1(x))$
(6) $Young(cell_2(x)) \leftarrow Change(x, cell_2(x))$
(7) $Mature(grow(x)) \leftarrow Change(x, grow(x))$
(8) $Warm(grow(x)) \vee Cold(grow(x)) \leftarrow Change(x, grow(x))$
(9) $Warm(y) \leftarrow Warm(x), Change(x, y), not\ Cold(y)$
(10) $Cold(y) \leftarrow Cold(x), Change(x, y), not\ Warm(y)$
(11) $\leftarrow Cold(x), Warm(x)$
(12) $Young(b) \leftarrow$
(13) $Warm(b) \leftarrow$

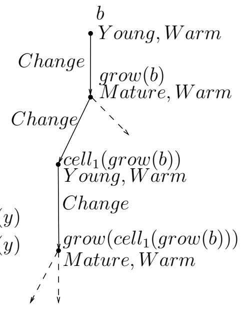

Fig. 1. Example: Evolution of a Cell

3 𝔽𝔻ℕℂ Programs

We now introduce the class 𝔽𝔻ℕℂ of logic programs with function symbols. The syntactic restrictions that are applied are to ensure the decidability of the formalism. As we shall see, 𝔽𝔻ℕℂ programs can have infinitely many possibly infinite stable models. In this section we analyze the model-theoretic properties of the formalism and introduce a method for finite representation of those possibly infinite stable models of a program.

Definition 1. *An 𝔽𝔻ℕℂ program is a finite disjunctive logic program whose rules are of the following forms:*

$$\begin{array}{lrcl}
(R1) & A_1(x) \vee \ldots \vee A_n(x) & \leftarrow & (not)B_0(x), \ldots, (not)B_l(x) \\
(R2) & R_1(x,y) \vee \ldots \vee R_n(x,y) & \leftarrow & (not)P_0(x,y), \ldots, (not)P_l(x,y) \\
(R3) & R_1(x, f_1(x)) \vee \ldots \vee R_n(x, f_n(x)) & \leftarrow & (not)P_0(x, g_0(x)), \ldots, (not)P_l(x, g_l(x)) \\
(R4) & A_1(y) \vee \ldots \vee A_n(y) & \leftarrow & (not)B_0(Z_0), \ldots, (not)B_l(Z_l), R(x,y) \\
(R5) & A_1(f(x)) \vee \ldots \vee A_n(f(x)) & \leftarrow & (not)B_0(W_0), \ldots, (not)B_l(W_l), R(x, f(x)) \\
(R6) & R_1(x, f_1(x)) \vee \ldots \vee R_n(x, f_k(x)) & \leftarrow & (not)B_0(x), \ldots, (not)B_l(x) \\
(R7) & C_1(\boldsymbol{c_1}) \vee \ldots \vee C_n(\boldsymbol{c_n}) & \leftarrow & (not)D_1(\boldsymbol{d_1}), \ldots, (not)D_l(\boldsymbol{d_l}),
\end{array}$$

where $n, l \geq 0$, each $Z_i \in \{x, y\}$, $W_i \in \{x, f(x)\}$, and each $\boldsymbol{c_i}$, $\boldsymbol{d_i}$ is a tuple of constants of arity ≤ 2. Each rule r is safe*, i.e., each of its variables occurs in* $\mathsf{body}^+(r)$*. Moreover, at least one rule is of type (R7) and is a fact.*

The fragments obtained from 𝔽𝔻ℕℂ by disallowing disjunction, constraints or negative literals are denoted by respectively removing 𝔻, ℂ or ℕ from "𝔽𝔻ℕℂ".

The structure of the rules in 𝔽𝔻ℕℂ syntax, the availability of non-monotonic negation and function symbols allow us to represent possibly infinite processes in a rather natural way. We provide here an example from the biology domain.

Example 1. The 𝔽𝔻ℕℂ program P in Figure 1 represents the evolution of a cell; its growth and splitting into two cells. The rules (1)-(4) describe changes of a cell. If it is warm, a young cell will grow and a mature cell will split into two cells; any cell dies if it is cold. The rules (5)-(7) determine whether a cell is young or mature. The rules (8)-(11) state the knowledge about the temperature. During the growth (which takes longer time), it might alter, while in the other changes (which take short time), it stays the same; the latter is expressed by inertia rules (9) and (10). Finally, (12) and (13) are the initialization facts.

It is easy to see that P is consistent. In fact, it has infinitely many stable models, corresponding to the possible evolutions of the initial situation. It might have finite and infinite stable models, as cell splitting might go on forever. The piece of the stable model that is depicted represents a development where the temperature does not change during the growth of b and its child. Another stable model is $\{Young(b), Warm(b), Change(b, grow(b)), Cold(grow(b)), Mature(grow(b)),$ $Change(grow(b),\ die(grow(b)))\}$ which corresponds to the situation that the temperature changes and the bacterium dies.

The brave query $\exists x.Cold(x)$ evaluates to true, while it is not the case for the brave query $Change(b, die(b))$. Note that the query whether there is some evolution in which bacteria never die is expressed by adding the constraint $\leftarrow$ $Change(x, die(x))$ and asking whether the resulting program is consistent.

Example 2. $\mathbb{FDNC}$ is well-suited to encode action domain descriptions in transition-based action formalisms which support incomplete states and nondeterministic action effects, like C [7], $\mathcal{K}$ [3], or (propositional) situation calculus.

We outline the elements for a possible such encoding. A unary predicate $\mathit{Sit}(x)$ encodes the situation in which the domain is, where the initial situation is given by $\mathit{Sit}(\mathit{init}) \leftarrow$. State descriptions are in terms of unary fluent predicates $F(x)$, which intuitively means that F is true in the state associated with x.

A predicate $\mathit{Trans}(x, y)$ describes the transition from situation x to the next situation y; for this, the rule $\mathit{Sit}(y) \leftarrow \mathit{Trans}(x, y)$ is included. Transitions are due to actions α_1, ..., α_k, which can be represented using function symbols f_{α_1}, ..., f_{α_k}. A rule $\mathit{Trans}(x, f_{\alpha_1}(x)) \vee \cdots \vee \mathit{Trans}(x, f_{\alpha_k}(x)) \leftarrow \mathit{Sit}(x)$ may describe the action execution, while the constraints $\leftarrow \mathit{Trans}(x, f_{\alpha_i}(x)), \mathit{Trans}(x, f_{\alpha_j}(x))$, for all different i and j ensure that actions are not concurrent.

Action effects during a transition can be stated by rules, while executability conditions for actions can be stated by constraints; in particular, inertia for fluent F can be expressed using the rule $F(y) \leftarrow F(x), \mathit{Trans}(x, y), \mathit{not}\ \mathit{neg\_F}(y)$ where $\mathit{neg\_F}(x)$ is a predicate for the complement of F.

Using these elements, $\mathbb{FDNC}$ may be used to represent a number of actions domains from the literature, e.g., the Yale Shooting, Bomb in the Toilet, and others cf. [3]; in fact, usually $\mathbb{FN}$ is already convenient.

Example 1 shows that in presence of function symbols, an $\mathbb{FDNC}$ program may have infinite stable models. We present in the sequel a method to finitely represent the possibly infinite stable models. To this end, we first provide a semantic characterization of the stable models of an $\mathbb{FDNC}$ program.

3.1 Semantic Characterization of Stable Models

Like many decidable logics, including Description Logics, $\mathbb{FDNC}$ programs enjoy a *forest-shaped model property.* A stable model of an $\mathbb{FDNC}$ program can be viewed as a graph and a set of trees rooted at each of the nodes in the graph.

Proposition 1. *An interpretation I is* forest-shaped*, if the following hold:*

(a) Each atom in I is either unary or binary. Additionally, each binary atom is of the form $R(c,d)$ or $R(t,f(t))$, where c, d are constants, and t is a term.
(b) If $A \in I$ is an atom with a term of the form $f(t)$ occurring as an argument, then for some binary predicate symbol R, $R(t,f(t)) \in I$.

If H is an arbitrary interpretation for an $\mathbb{FDNC}$ program P and $J \in MM(P^H)$, then J is forest-shaped. Therefore, every $J \in SM(P)$ is forest-shaped.

The methods that we present in this paper are aimed at providing the decidability results together with the worst-case optimal algorithms for $\mathbb{FDNC}$. We note, however, that the decidability of the reasoning tasks discussed in this paper can be inferred from the results in [5]. The technique in [5] shows how the stable model semantics for the disjunctive logic programs with functions symbols can be expressed by formulae in second-order logic, where the minimality of models is enforced by second-order quantifiers. Due to the forest-shaped model property one can express the semantics of $\mathbb{FDNC}$ programs in monadic second-order logic over trees *SkS* which is know to be decidable (see [14] for a related encoding). Unfortunately, optimal algorithms for such encodings are not apparent.[1]

The semantic characterization, and the reasoning methods later on, follow an intuition that stable models for an $\mathbb{FDNC}$ program P can be constructed by the iterative computation of stable models of *local programs.* During the construction, local programs are obtained "on the fly" by taking certain finite subsets of $\mathsf{Ground}(P)$ and adding facts (*states*) obtained in the previous iteration.

For the rest of Section 3, we assume that P is an arbitrary $\mathbb{FDNC}$ program. For the convenience of presentation, for a term t and a set of atoms I, we write $t \hat{\in} I$, if there exists an atom in I with t as its argument.

Definition 2. *Let t be a term. A* state *of t is an arbitrary set U^t containing only unary atoms ground with t (i.e., with t as the argument); the superscript t will be dropped if t is not of particular interest. For a set of atoms I and a term $t \hat{\in} I$, we denote by $\mathsf{st}(I,t)$ the state of t in I, i.e., the set $\{A(t) \mid A(t) \in I\}$.*

For a one-variable rule r in $\mathbb{FDNC}$ syntax, let $r_{\downarrow t}$ denote the rule obtained by substituting every occurrence of the variable in r with a term t. Without loss of generality, we assume that in a two-variable rule, i.e., a rule of type (R2) or (R4), the tuple of variables in binary atoms is always $\langle x, y\rangle$. For such a rule r,

[1] Via an encoding into *SkS*, one can show the decidability of $\mathbb{FDNC}$ extended with *inverse* rules of the form $R(y,x) \leftarrow P(x,y)$, which, together with the rules of type (R4), allow for bidirectionality of information-passing. Due to more involved minimality-testing, our techniques cannot be extended easily to handle inverse rules.

let $r_{\downarrow s,t}$ denote the rule obtained by substituting every occurrence of x with a term s and every occurrence of y with a term t.

Definition 3. *Let U^t be a state. The* local program $P(U^t)$ *is the smallest program containing the following rules:*

- *$A(t) \leftarrow$, for each $A(t) \in U^t$,*
- *$r_{\downarrow t}$, for each $r \in P$ of type (R3), (R5), or (R6),*
- *$r_{\downarrow t,f(t)}$, for each $r \in P$ of type (R2) or (R4) and function symbol f of P, and*
- *$r_{\downarrow f(t)}$, for each $r \in P$ of type (R1) and function symbol f of P.*

Suppose I is a forest-shaped interpretation for P, $t \hat{\in} I$, and U is the state of t in I, i.e., $U = \mathsf{st}(I, t)$. Intuitively, the stable models of $P(U)$ define the set of possible immediate successor structures for t in I. In other words, if I is a stable model of P, then I must contain a stable model of $P(U)$. Stable models of local programs have a simple structural property, captured by the notion of *knots*.

Definition 4. *(Knots) A* knot with a root term t *is a set of atoms K such that (i) each atom in K has form $A(t)$, $R(t, f(t))$, or $A(f(t))$ where A, R, and f are arbitrary, and (ii) for each term $f(t) \hat{\in} K$, there exists $R(t, f(t)) \in K$ (connectedness). Let $\mathsf{succ}(K)$ denote the set of all terms $f(t) \hat{\in} K$.*

A knot with a root term t can be viewed as a labeled tree of depth at most 1, where $\mathsf{succ}(K)$ are the leaf nodes. The nodes are labeled with unary predicate symbols, while the edges are labeled with binary predicate symbols. Note that $\emptyset$ is a knot whose root term can be arbitrary.

It is easy to see that due to the structure of local programs, their stable models satisfy the conditions in the definition of knots, and therefore are knots. On the other hand, knots are also the structures that appear in the trees of the forest-shaped interpretations. To "extract" a knot occurring in a forest-shaped interpretation, the following will be helpful.

For a term t, let $\mathcal{HB}_t$ denote the set of all atoms that can be built from unary and binary predicate symbols using t and terms of the form $f(t)$. For any forest-shaped interpretation I of P and $t \hat{\in} I$, the set $K := I \cap \mathcal{HB}_t$ is a knot.

A knot K with a root term t is *over (the signature of) P*, if each predicate and function symbol occurring in K also occurs in P (t need not be from $\mathcal{HU}^P$).

The following notion is central. The introduced *stable* knots are self-contained model building blocks for $\mathbb{FDNC}$ programs.

Definition 5. *(Stable Knot) Let K be a knot with a root term t and $U^t = \mathsf{st}(K, t)$. Then K is* stable *w.r.t. the program P iff $K \in SM(P(U^t))$.*

Intuitively, stable knots encode an assumption and a solution. Suppose a knot K with a root term t is stable w.r.t. P. Moreover, suppose t occurs in a forest-shaped interpretation I for P, as a "leaf node", i.e., there are no atoms of the form $R(t, f(t))$ in I. Intuitively, if the states of t in I and K coincide, i.e., $\mathsf{st}(I, t) = \mathsf{st}(K, t)$, then K becomes an eligible set of atoms that can be introduced in I to give t the necessary successors.

After introducing the necessary notions for dealing with the tree-part of forest-shaped interpretations, we deal with the graph part.

Definition 6. *By P^{G} we denote the program* $\mathsf{Ground}(P')$*, where P' is obtained from P by removing all the rules containing function symbols.*[2]

The following theorem characterizes the stable models of P. For an interpretation I, let I^c be the set of all atoms $A(\boldsymbol{c}) \in I$ such that $\boldsymbol{c}$ is a tuple of constants.

Theorem 1. *Let I be an interpretation for P. Then I is a stable model of P iff I is a forest-shaped interpretation such that (i) I^c is a stable model of P^{G}, and (ii) for each term $t \hat{\in} I$, $I \cap \mathcal{HB}_t$ is a knot that is stable w.r.t. P.*

3.2 Finite Representation of Stable Models

The semantic characterization of stable models for $\mathbb{FDNC}$ programs allows to view stable models as being constructed of knots, where each of them is stable. Next, we show that Theorem 1 allows us to provide a finite representation of those stable models. Roughly, it is based on the observation that although infinitely many knots might occur in some stable model of a program, only finitely many of them are non-isomorphic modulo the root term.

Definition 7. *Let K be a knot with a root term t. By $K_{\downarrow u}$ we denote the knot obtained from K by replacing each occurrence of t in K with a term u.*

Indeed, if the program P has an infinite stable model I, then set of knots $L := \{(I \cap \mathcal{HB}_t) \mid t \hat{\in} I\}$ is infinite. However, for a fixed term t, the set $L' := \{K_{\downarrow t} \mid K \in L\}$ is finite due to the fact that there are only finitely many knots with the root term t over the signature of P. Intuitively, if we view t as a variable, then each $K \in L$ can be viewed as an instance of some knot in L'.

To talk about sets of knots with common root term, we assume a special constant $\mathbf{x}$ not occurring in $\mathbb{FDNC}$ programs. We say a set of knots is $\mathbf{x}$*-grounded*, if it contains only knots with root term $\mathbf{x}$. The following notion lets us collect the knots occurring in a stable model and abstract them by substituting with $\mathbf{x}$.

Definition 8. *(Scanning) Let I be a forest-shaped interpretation for P. We define the set of* $\mathbf{x}$*-grounded knots* $\mathbb{K}(I) := \{(I \cap \mathcal{HB}_t)_{\downarrow \mathbf{x}} \mid t \hat{\in} I\}$.

In the following we show that $\mathbf{x}$-grounded sets of knots can be used to represent the stable models of an $\mathbb{FDNC}$ program. First, we observe that the stability of a knot is preserved under substitutions.

Proposition 2. *If K is a knot that is stable w.r.t. P, and u is an arbitrary term, then $K_{\downarrow u}$ is stable w.r.t. P.*

We introduce the notion of *founded* sets of $\mathbf{x}$-grounded knots. The intention is to capture the properties of the set $\mathbb{K}(I)$ when I is a stable model of P. To this end, we need a notion of *state equivalence* as a counterpart for substitutions in knots. Formally, states U^t and V^s are *equivalent* (in symbols, $U^t \approx V^s$), if $U^t = \{A(t) \mid A(s) \in V^s\}$, i.e., in both states terms satisfy the same unary predicates.

[2] Note that P^{G} is finite since its Herbrand universe contains only the constants of P.

Definition 9. *Let $S \neq \emptyset$ be a set of states. A set L of $\mathbf{x}$-grounded knots that are stable w.r.t. the program P is* founded *w.r.t. P and S, if the following hold:*

1. *For each $U \in S$, there exists $K \in L$ such that $U \approx \mathsf{st}(K, \mathbf{x})$.*
2. *For each $K \in L$, the following hold:*
   a. *for each $s \in \mathsf{succ}(K)$, there exists $K' \in L$ s.t. $\mathsf{st}(K, s) \approx \mathsf{st}(K', \mathbf{x})$, and*
   b. *there exists a sequence $\langle K_0, \ldots, K_n \rangle$ of knots in L such that:*
      - *$K_n = K$,*
      - *K_0 is such that $\mathsf{st}(K, \mathbf{x}) \approx U$ for some $U \in S$, and*
      - *for each $0 \leq i < n$, there exists $s \in \mathsf{succ}(K_i)$ s.t. $\mathsf{st}(K_i, s) \approx \mathsf{st}(K_{i+1}, \mathbf{x})$.*

For an interpretation I, let $S(I)$ denote the set of states of constants occurring in I, i.e., $S(I) := \{\mathsf{st}(I, c) \mid c \hat{\in} I \text{ is a constant}\}$. The following is easy to verify.

Proposition 3. *If I is a stable model of P, then $\mathbb{K}(I)$ is a set of knots that is founded w.r.t. P and $S(I^c)$.*

In what follows we provide a construction of stable models out of knots in a founded set. Moreover, we show that for a given consistent program there exists a founded set of knots that captures all the stable models.

Generating Stable Models out of Knots. Before describing the construction of forest-shaped interpretations, we first state the construction of trees, which are represented in the standard way by prefix-closed sets of words. For a sequence of elements $p = [e_1, \ldots, e_n]$, let $\tau(p)$ denote the last element e_n, and $[p|e_{n+1}]$ denote the sequence $[e_1, \ldots, e_n, e_{n+1}]$.

Definition 10. *(Tree Construction) Let L be a set of knots that is founded w.r.t. P and a set of states S, and let U^t be a state such that $U^t \approx V$, for some $V \in S$. A set T of sequences, where each element in a sequence is a tuple of a knot and a term, is called a* tree induced by L starting at U^t, *if the following hold:*

(a) *$[\langle K, t \rangle] \in T$, where $K \in L$ is s.t. $\mathsf{st}(K, \mathbf{x}) \approx U^t$.*
(b) *If there exists $p \in T$ with $\tau(p) = \langle K, t \rangle$ and $f(\mathbf{x}) \in \mathsf{succ}(K)$, then there exists $[p|\langle K', f(t) \rangle] \in T$, where K' is a knot in L s.t. $\mathsf{st}(K, f(\mathbf{x})) \approx \mathsf{st}(K', \mathbf{x})$.*
(c) *T is minimal, i.e., each $T' \subset T$ violates (a) or (b).*

We state the transformation of trees into Herbrand interpretations.

Definition 11. *Let T be a tree induced by a founded set of knots L starting at some state. We define the set of atoms $T_{\downarrow} := \{K_{\downarrow t} \mid p \in T \text{ with } \tau(p) = \langle K, t \rangle\}$.*

We generalize the construction of trees to forest-shaped interpretations.

Definition 12. *(Forest Construction) Let G be a set of atoms ground with the constants of P only, and L be a set of knots founded w.r.t. P and a set of states $S \supseteq S(G)$. Then $\mathcal{F}(G, L)$ is the largest set of forest-shaped interpretations*

$$I = G \cup (T^{c_1})_{\downarrow} \cup \ldots \cup (T^{c_n})_{\downarrow},$$

where $\{c_1, \ldots, c_n\}$ is the set of all constants occurring in G and each T^{c_i} a tree induced by L starting at $\mathsf{st}(G, c_i)$.

$\mathcal{F}(G,L)$ represents all the forest-shaped interpretations that can be build from G by attaching, for each of the constants, a tree induced by L.

Theorem 2. *If $G \in SM(P^{\mathsf{G}})$, L is a set of knots that is founded w.r.t. P and some $S \supseteq S(G)$, then $\mathcal{F}(G,L) \neq \emptyset$ and each $I \in \mathcal{F}(G,L)$ is a stable model of P.*

Proof. Indeed, $\mathcal{F}(G,L) \neq \emptyset$ due to foundedness of L. Assume some $I \in \mathcal{F}(G,L)$. Each $K \in L$ is stable w.r.t. P. Then due to Proposition 2, for each term $t \hat{\in} I$, $I \cap \mathcal{HB}_t$ is a knot that is stable w.r.t. P. Keeping in mind that $G \in SM(P^{\mathsf{G}})$, Theorem 1 implies that I is a stable model of P.

We showed that stable model existence can be proved by checking that some founded set of knots exists. As we see next, the properties of founded sets of knots imply that we can obtain a set capturing all the stable models of a program.

Capturing Stable Models. We define the set of states that occur in the stable models of P^{G} as $S(P) := \{\mathsf{st}(G,c) \mid G \in SM(P^{\mathsf{G}}) \wedge c \hat{\in} G\}$.

Definition 13. *By $\mathbb{K}_P$ we denote the smallest set of knots which contains every set of knots L that is founded w.r.t. P and some $S \subseteq S(P)$.*

Proposition 4. *For the program P, the following hold:*

(a) If $\mathbb{K}_P \neq \emptyset$, then $\mathbb{K}_P$ is founded w.r.t. P and some $S \subseteq S(P)$.
(b) If L is a set of knots that is founded w.r.t. P and some $S \subseteq S(P)$, then $\mathbb{K}_P$ is founded w.r.t. P and some $S' \supseteq S$.
(c) Each $L \supset \mathbb{K}_P$ is not founded w.r.t. P and any $S \subseteq S(P)$.

Proof. The claim follows from the following property: if L_1 and L_2 are two sets of knots founded w.r.t. P and, respectively, sets of states S_1 and S_2, then $L_1 \cup L_2$ is founded w.r.t. P and $S_1 \cup S_2$.

It is easy to verify that a stable model I can be reconstructed out of knots in $\mathbb{K}(I)$. Naturally, the same holds for any superset of $\mathbb{K}(I)$ satisfying Definition 9.

Proposition 5. *If I is a stable model of P, then $I \in \mathcal{F}(I^c, L)$ for each set of knots $L \supseteq \mathbb{K}(I)$ s.t. L is founded w.r.t. P and some set of states $S \supseteq S(I^c)$.*

The following will be helpful.

Definition 14. *We say $\mathbb{K}_P$ is* compatible *with a set of states S, if for each state $U \in S$, there exists $K \in \mathbb{K}_P$ s.t. $U \approx \mathsf{st}(K,\mathbf{x})$.*

The crucial property of $\mathbb{K}_P$ is that it captures the tree-structures of all the stable models of P. Together with the stable models of P^{G}, it represents the stable models of P.

Theorem 3. *Let I be an interpretation for P. Then, $I \in SM(P)$ iff $I \in \mathcal{F}(G, \mathbb{K}_P)$, for some $G \in SM(P^{\mathsf{G}})$ s.t. $\mathbb{K}_P$ is compatible with $S(G)$.*

Table 1. Complexity of $\mathbb{FDNC}$ and Fragments (Completeness Results)

Fragments	Consistency	$P \models_b A(\boldsymbol{t})$	$P \models_b \exists \boldsymbol{x}.A(\boldsymbol{x})$
$\mathbb{F}$	Trivial	P	PSpace
$\mathbb{FD}$	Trivial	Σ_2^P	PSpace
$\mathbb{FC}$	PSpace	PSpace	PSpace
$\mathbb{FDC}$, $\mathbb{FN}$, $\mathbb{FDNC}$	ExpTime	ExpTime	ExpTime

Proof. If $I \in SM(P)$, then, by Prop. 3, $\mathbb{K}(I)$ is founded w.r.t. P and $S(I^c)$. By definition, $\mathbb{K}(I) \subseteq \mathbb{K}_P$. By Prop. 4, $\mathbb{K}_P$ is founded w.r.t. P and some $S \supseteq S(I^c)$. By Proposition 5, $I \in \mathcal{F}(I^c, \mathbb{K}_P)$. The other direction is proved by Theorem 2.

We have obtained a finite representation of stable model of an $\mathbb{FDNC}$ program. Indeed, all of its stable models can be generated out of some stable model of P^{G} and a set of knots $\mathbb{K}_P$.

4 Reasoning and Complexity

The algorithms for consistency check and brave entailment are based on computing $\mathbb{K}_P$. The complexity of these tasks for $\mathbb{FDNC}$ and its fragments is compactly summarized in Table 1. For space reasons, we focus here on $\mathbb{FDNC}$ and briefly discuss the other fragments at the end of this section.

Deriving the Set $\mathbb{K}_P$*.* To derive $\mathbb{K}_P$, we proceed in two phases. In the first phase, we generate the set of knots $\mathsf{All}(P)$ that surely contains $\mathbb{K}_P$. In the second phase, we remove knots from it to ensure that it satisfies Definition 13.

To ease presentation, for a knot set L, let $\mathsf{states}(L) := \{\mathsf{st}(K,s) \mid K \in L, s \in \mathsf{succ}(K)\}$ be the set of all states of the successor terms of knots in L.

Definition 15. *For an* $\mathbb{FDNC}$ *program* P*, let* $\mathsf{All}(P)$ *be the smallest set of* $\mathbf{x}$*-grounded knots obeying the following conditions:*

a) If $U \in S(P)$ *and* $K \in SM(P(U))$*, then* $K_{\downarrow \mathbf{x}} \in \mathsf{All}(P)$*.*
b) If $U \in \mathsf{states}(\mathsf{All}(P))$ *and* $K \in SM(P(U))$*, then* $K_{\downarrow \mathbf{x}} \in \mathsf{All}(P)$*.*

By construction, $\mathsf{All}(P)$ contains each set of knots which is founded w.r.t. P and some set of states $S \subseteq S(P)$. The problem is that $\mathsf{All}(P)$ might contain a knot K such that some $s \in \mathsf{succ}(K)$ has no potential successor knot (see (2.a) in Definition 9). Such knots should be removed from $\mathsf{All}(P)$. In turn, such removal might leave some knots in $\mathsf{All}(P)$ without a potential predecessor (see (2.b) in Definition 9). The second phase deals with this problem.

Definition 16. *For any set of* $\mathbf{x}$*-grounded knots* L *and set of states* S*,* $\mathsf{reach}(L,S)$ *is the smallest set of knots such that:*

a) if $U \in S$*,* $K \in L$ *and* $U \approx \mathsf{st}(K,\mathbf{x})$*, then* $K \in \mathsf{reach}(L,S)$*, and*
b) if $U \in \mathsf{states}(\mathsf{reach}(L,S))$*,* $K \in L$ *and* $U \approx \mathsf{st}(K,\mathbf{x})$*, then* $K \in \mathsf{reach}(L,S)$*.*

```
Algorithm Knots
Input: $\mathbb{FDNC}$ program $P$
Output: $\mathbb{K}_P$
repeat
  $L := \mathsf{All}(P)$; $S := S(P)$; $L^{aux} := L$;
  for each $K \in L$ and $s \in \mathsf{succ}(K)$ do
     if not $\exists K' \in L$ s.t. $\mathsf{st}(K, s) \approx \mathsf{st}(K', \mathbf{x})$ then $L := L \setminus \{K\}$;
  $L := \mathsf{reach}(L, S)$
until $L^{aux} = L$
return $L$
```

Fig. 2. Algorithm for computing $\mathbb{K}_P$ of an $\mathbb{FDNC}$ program

Intuitively, $\mathsf{reach}(L, S)$ are the knots in L reachable from the states in S. Thus, if $\mathsf{reach}(L, S) = L$, then L fulfills the condition (2.b) of Definition 9 to be founded w.r.t. S.

The cleaning of $\mathsf{All}(P)$ involves removing each knot violating (2.a) or (2.b) in Definition 9. Figure 2 shows an algorithm for computing $\mathbb{K}_P$ which elaborates on this.

Soundness and Completeness. We must verify that *Knots(P)* satisfies the condition in Definition 13, i.e., *Knots(P)* is the single (set-inclusion) minimal set which contains each set L of knots that is founded w.r.t. P and some $S \subseteq S(P)$.

Indeed, $L \subseteq \mathsf{All}(P)$ by construction. In the computation of *Knots(P)* no knot in L can be removed from $\mathsf{All}(P)$, i.e., $L \subseteq Knots(P)$. Suppose *Knots(P)* is not minimal. Hence, some $N \subset Knots(P)$ contains every knot set L which is founded w.r.t. P and some $S \subseteq S(P)$. Then $Knots(P)$ must be nonempty, and it holds that $Knots(P)$ is founded w.r.t. P and some $S \subseteq S(P)$. Roughly, this is because the algorithm ensures that every knot in $Knots(P)$ is stable w.r.t. P, has proper successors to satisfy (2.a) in Definition 9, and has a proper sequence of predecessors to satisfy (2.b) reaching a state in $S(P)$. By assumption on N and foundedness of $Knots(P)$, we have $Knots(P) \subseteq N$. This, however, contradicts $N \subset Knots(P)$. Thus $Knots(P)$ satisfies Definition 13, i.e., $Knots(P) = \mathbb{K}_P$.

Complexity. The procedure $Knots(P)$ runs in time single exponential in the size of P. The claim follows from the following observations:

- The number of $\mathbf{x}$-grounded knots over P, max, is bounded by single exponential in the size of P; more precisely, $max \leq 2^{n+k\cdot(n+m)}$, when P has k function, n unary, and m binary predicate symbols.
- Computing $\mathsf{All}(P)$ requires adding at most max $\mathbf{x}$-grounded knots. Each such knot has polynomial size and its stability is verifiable using an NP^{NP} oracle. Thus, $\mathsf{All}(P)$ is computable in time single exponential in the size of P.
- Computing $\mathsf{reach}(L, S)$ is polynomial in the combined size of L and S.
- The size of $S(P)$ is bounded by a single exponential in the size of P.
- $Knots(P)$ runs in time that is polynomial in the size of $\mathsf{All}(P)$ and $S(P)$.

Consistency Check. Theorems 2 and 3 imply the following characterization.

Theorem 4. *An $\mathbb{FDNC}$ program P is consistent iff $\mathbb{K}_P$ is compatible w.r.t. $S(G)$, for some $G \in SM(P^{\mathsf{G}})$.*

Compatibility of $\mathbb{K}_P$ w.r.t. $S(G)$, for $G \in SM(P^{\mathsf{G}})$, is decidable in time polynomial in $n + m$, where m is the size of $\mathbb{K}_P$ and n is the size of $SM(P^{\mathsf{G}})$. This is single exponential in the size of P, since both m and n are single exponential in the size of P. Since $SM(P^{\mathsf{G}})$ is computable in single exponential time, Theorem 4 implies that consistency checking in $\mathbb{FDNC}$ is feasible in single exponential time. In the full paper, by a reduction of satisfiability testing in the ExpTime-hard DL $\mathcal{ALC}$, we show that consistency check is ExpTime-hard already for $\mathbb{FDC}$. Thus, the algorithm emerging from Theorem 4 is worst-case optimal.

Theorem 5. *Deciding whether a given $\mathbb{FDNC}$ program is consistent, i.e., has some stable model, is* ExpTime*-complete.*

Brave Entailment. We can also exploit $\mathbb{K}_P$ for brave reasoning in $\mathbb{FDNC}$. We focus here on unary atomic queries; binary queries are easily reduced to this case. The idea is to perform "back-propagation" of unary predicate symbols in a founded set of knots. For a set of $\mathbf{x}$-grounded knots L, we call $K' \in L$ a *possible successor* of $K \in L$ if $\mathsf{st}(K', \mathbf{x}) \approx \mathsf{st}(K, s)$ for some $s \in \mathsf{succ}(K)$.

Definition 17. *Let L be a set of knots founded w.r.t. an $\mathbb{FDNC}$ program P and a set of states S. Let C be the set of unary predicate symbols occurring in P. By $\mathcal{E}_L$ we denote the smallest relation over $L \times C$ closed under the following rules:*

(a) if $K \in L$ and some $A(\mathbf{x}) \in K$, then $\langle K, A \rangle \in \mathcal{E}_L$, and
(b) if $K' \in L$ is a possible successor of $K \in L$ s.t. $\langle K', A \rangle \in \mathcal{E}_L$, then $\langle K, A \rangle \in \mathcal{E}_L$.

Intuitively, $\langle K, A \rangle \in \mathcal{E}_L$ means that starting from K a sequence of possible successor knots will eventually reach a knot containing $A(\mathbf{x})$. We have the following:

Theorem 6. *Let P be an $\mathbb{FDNC}$ program. Then, $P \models_b \exists x.A(x)$ iff ($\star$) for some $G \in SM(P^{\mathsf{G}})$, (a) $\mathbb{K}_P$ is compatible w.r.t. $S(G)$, and (b) there exist a constant c and $K \in \mathbb{K}_P$ such that $\mathsf{st}(G, c) \approx \mathsf{st}(K, \mathbf{x})$ and $\langle K, A \rangle \in \mathcal{E}_{\mathbb{K}_P}$.*

Condition ($\star$) is verifiable in time (single) exponential in the size of P. Indeed, computing $\mathcal{E}_{\mathbb{K}_P}$ requires time quadratic in the size of $\mathbb{K}_P$, or exponential in the size of P. Once $\mathbb{K}_P$, $\mathcal{E}_{\mathbb{K}_P}$, and $SM(P^{\mathsf{G}})$ are computed, the conditions in ($\star$) are verifiable in time polynomial in the combined size of $\mathbb{K}_P$, $\mathcal{E}_{\mathbb{K}_P}$, and $SM(P^{\mathsf{G}})$.

Via this algorithm, we obtain that brave reasoning for existential unary queries in $\mathbb{FDNC}$ is in ExpTime. In the full paper we show that the algorithm is worst-case optimal (by reducing consistency to brave entailment in $\mathbb{FDNC}$), and that this result extends to binary existential queries.

Theorem 7. *Deciding $P \models_b \exists \boldsymbol{x}.A(\boldsymbol{x})$ is* ExpTime*-complete for $\mathbb{FDNC}$.*

The method for deciding brave entailment of ground unary queries is based on an adaptation of the algorithm for the existential queries.

Definition 18. *Let $q = A(t)$ be a ground atom and L be a set of knots founded w.r.t. an $\mathbb{FDNC}$ program P and a set of states S. Let T be the set of subterms of the term t. Then $\mathcal{G}_L^q$ is the smallest relation over $L \times T$ such that:*

(a) if $K \in L$ and $A(\mathbf{x}) \in K$, then $\langle K, t\rangle \in \mathcal{G}_L^q$, and
(b) if there exist (i) $K \in L$ with $f(\mathbf{x}) \in \mathsf{succ}(K)$ and (ii) $K' \in L$ s.t. $\mathsf{st}(K, f(\mathbf{x})) \approx \mathsf{st}(K', \mathbf{x})$ and $\langle K', f(v)\rangle \in \mathcal{G}_L^q$, then $\langle K, v\rangle \in \mathcal{G}_L^q$.

Suppose we have a ground query $q = A(f(g(f(c))))$ and a knot K in L such that $\langle K, c\rangle \in \mathcal{G}_L^q$. Roughly, it means that we can construct a tree with root term c containing a node $f(g(f(c)))$ labeled with A. The following is easily verified.

Theorem 8. *For any $\mathbb{FDNC}$ program P and ground query q, $P \models_b q$ iff ($\star\star$) for some $G \in SM(P^{\mathsf{G}})$, (a) $\mathbb{K}_P$ is compatible w.r.t. $S(G)$, and (b) some $K \in \mathbb{K}_P$ exists s.t. $\mathsf{st}(G, c) \approx \mathsf{st}(K, \mathbf{x})$ and $\langle K, c\rangle \in \mathcal{G}_{\mathbb{K}_P}^q$, where c is the constant in q.*

By similar arguments as for existential queries, we can see that checking condition ($\star\star$) is feasible in time single exponential in the size of P. Note that computing $\mathcal{G}_{\mathbb{K}_P}^q$ requires time that is polynomial in the size of $\mathbb{K}_P$, or single exponential in the size of P. Once $\mathbb{K}_P$, $\mathcal{G}_{\mathbb{K}_P}^q$, and $SM(P^{\mathsf{G}})$ are computed, the conditions in ($\star\star$) can be verified in time polynomial in the combined size of $\mathbb{K}_P$, $\mathcal{G}_{\mathbb{K}_P}^q$, and $SM(P^{\mathsf{G}})$, each of which is single exponential in the size of P.

We thus have an algorithm for deciding $P \models_b A(t)$ in exponential time. The full paper shows that it is worst-case optimal, by providing an ExpTime-hardness result, and extends the result to binary ground queries.

Theorem 9. *Deciding $P \models_b A(\boldsymbol{t})$, for ground $\boldsymbol{t}$, is* ExpTime*-complete for* $\mathbb{FDNC}$.

The remaining entries in Table 1 are briefly explained as follows. $\mathbb{F}$ and $\mathbb{FD}$ programs are trivially consistent. For an $\mathbb{FC}$ program P, consistency can be decided by checking if each constant c in the single stable model G of P^{G} (if it exists) has a set of knots founded w.r.t. P and $\{\mathsf{st}(G, c)\}$. This can be refuted by nondeterministically constructing stepwise a sequence of at most exponentially many knots which leads to inconsistency. This is feasible in polynomial space, and since NPSpace = PSpace, consistency checking is in PSpace. Matching PSpace-hardness is shown by a generic Turing machine reduction, where a simple constraint of form $\leftarrow A(x)$ is sufficient to show the hardness.

The PSpace-hardness of $P \models_b \exists \boldsymbol{x}.A(\boldsymbol{x})$ for $\mathbb{F}$ is immediate from the fact $P \models_b \exists \boldsymbol{x}.A(\boldsymbol{x})$ holds iff $P \cup \{\leftarrow A(\boldsymbol{x})\}$ is inconsistent. The problem is in PSpace, since we can nondeterministically construct a sequence of at most exponentially many knots until A occurs. For $\mathbb{FD}$, this can be decided similarly; for $\mathbb{FC}$, an additional consistency check has to be made, but polynomial space is sufficient.

In case of ground entailment $P \models_b A(\boldsymbol{t})$, membership of $A(\boldsymbol{t})$ in the single stable model of an $\mathbb{F}$ or $\mathbb{FC}$ program is witnessed by a sequence of knots that is fully determined by $\boldsymbol{t}$ and computable in polynomial time. For $\mathbb{FC}$, the consistency check of P remains to be done. In case of $\mathbb{FD}$, we still need to guess knots, guided by $\boldsymbol{t}$, and verify their stability; this is feasible in Σ_2^P. Matching Σ_2^P-hardness follows from propositional logic programs [4].

Finally, ExpTime-completeness of the reasoning tasks for $\mathbb{FN}$ is proved by a polynomial reduction of consistency check in $\mathbb{FDC}$ to consistency check in $\mathbb{FN}$.

5 Related Work

Our $\mathbb{FN}$ programs are decidable *Finitely Recursive Programs (FRPs)* [2,1], which are normal logic programs P with function symbols where in the grounding of P, each atom depends only on finitely many atoms; disjunction and constraints are not allowed. For FRPs, inconsistency checking is r.e.-complete and brave ground entailment is co-r.e.-complete [1]; for $\mathbb{FN}$ and our full class $\mathbb{FDNC}$, which implicitly obeys the condition of FRPs, these problems are ExpTime-complete. On the other hand, $\mathbb{FN}$ is not a subclass of the *Finitary Programs (FPs)* [2], which are those RFPs in whose grounding only finitely many atoms occur in odd cycles. For FPs, consistency checking is decidable, and brave and cautious entailment are decidable for ground queries but r.e.-complete for existential atomic queries. Note that for $\mathbb{FN}$, all these problems are decidable in exponential time. Finally, the explicit syntax of $\mathbb{FN}$ and our other fragments of $\mathbb{FDNC}$ allows to effectively recognize such programs. FRPs and FPs, instead, suffer the undecidability of the conditions that define them, i.e., FRPs and FPs cannot be effectively recognized.

Related to our work are *Local Extended Conceptual Logic Programs (LECLPs)* [9], which evolved from [8]. These programs are function-free but have answer sets over *open domains*, i.e., of the grounding of a program with any superset of its constants. LECLPs are syntactically restricted to ensure the forest-shape model property of answer sets. Deciding consistency of an LECLP P is feasible in nondeterministic triple exponential time, as one can ground P with double exponentially many constants in the size of P, and then use standard ASP. For $\mathbb{FDNC}$, deciding the consistency is ExpTime-complete and thus less complex.

Comparing the expressiveness of LECLPs and $\mathbb{FDNC}$ is intricated due the different settings. At least, both formalisms can encode certain description logics (e.g., $\mathcal{ALC}$). LECLPs may be more expressive than $\mathbb{FDNC}$ programs, since the expressive DL $\mathcal{ALCHOQ}$ is reducible to satisfiability in LECLPs. On the other hand, LECLPs undermine the general intuition behind minimal model semantics of logic programs. So-called *free rules* of the form $p(x) \vee not\ p(x) \leftarrow;$ allow to unfoundedly add atoms in an answer set. $\mathbb{FDNC}$, instead, has no free rules, and each atom in a stable model of P must be justified from the very facts of P.

6 Discussion and Conclusion

In line with efforts to pave the way for effective Answer Set Programming engines with function symbols [2,1], we presented $\mathbb{FDNC}$ programs as a decidable class of disjunctive logic programs with function symbols under stable model semantics. They are a tool for knowledge representation and reasoning for some applications involving infinite processes and objects, like evolving action domains. From our results on consistency checking and brave entailment of ground and existential atomic queries q, one can easily determine the complexity of cautious entailment

$P \models_c q$, i.e., whether q is true in all stable models of P. The results in Table 1 for brave entailment carry over to cautious entailment except for $\mathbb{FD}$; here, $P \models_c A(\boldsymbol{t})$ is coNP-complete and $P \models_c \exists \boldsymbol{x}.A(\boldsymbol{x})$ is ExpTime-complete. Intuitively, the former is because minimality of models is irrelevant for inference of an atom $A(\boldsymbol{t})$, and the latter because consistency checking with constraints $\leftarrow B(\boldsymbol{x})$ can be reduced to cautious inference with rules $A(\boldsymbol{x}) \leftarrow B(\boldsymbol{x})$.

$\mathbb{FDNC}$ programs can be easily extended with strong negation $\neg p(\boldsymbol{x})$ [6], which can be expressed in the language as usual (view $\neg p$ as a predicate symbol and add constraints $\leftarrow p(\boldsymbol{x}), \neg p(\boldsymbol{x})$). Implementation of $\mathbb{FDNC}$ programs is another subject of future work. To this aim, recent extensions of the DLV system like DLVHEX (`http://con.fusion.at/dlvhex/`) might be exploited.

Acknowledgments. We thank a reviewer for pointing out the decidability of $\mathbb{FDNC}$ (even with inverse rules) via monadic second-order logic over trees.

References

1. Baselice, S., Bonatti, P.A., Criscuolo, G.: On finitely recursive programs. In: ICLP (accepted for publication, 2007)
2. Bonatti, P.A.: Reasoning with infinite stable models. Artif. Intell. 156(1), 75–111 (2004)
3. Eiter, T., Faber, W., Leone, N., Pfeifer, G., Polleres, A.: A Logic Programming Approach to Knowledge-State Planning: Semantics and Complexity. ACM Transactions on Computational Logic 5(2), 206–263 (2004)
4. Eiter, T., Gottlob, G.: On the Computational Cost of Disjunctive Logic Programming: Propositional Case. Annals of Mathematics and Artificial Intelligence 15(3/4), 289–323 (1995)
5. Eiter, T., Gottlob, G.: Expressiveness of stable model semantics for disjuncitve logic programs with functions. J. Log. Program. 33(2), 167–178 (1997)
6. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Comput. 9(3/4), 365–386 (1991)
7. Giunchiglia, E., Lifschitz, V.: An Action Language Based on Causal Explanation: Preliminary Report. In: AAAI 1998. Proceedings of the Fifteenth National Conference on Artificial Intelligence, pp. 623–630 (1998)
8. Heymans, S.: Decidable Open Answer Set Programming. PhD thesis, Theoretical Computer Science Lab (TINF), Department of Computer Science, Vrije Universiteit Brussel, Pleinlaan 2, B1050 Brussel, Belgium (February 2006)
9. Heymans, S., Nieuwenborgh, D.V., Vermeir, D.: Nonmonotonic ontological and rule-based reasoning with extended conceptual logic programs. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, pp. 392–407. Springer, Heidelberg (2005)
10. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM Transactions on Computational Logic 7(3), 499–562 (2006)
11. Marek, V.W., Remmel, J.B.: On the expressibility of stable logic programming. In: LPNMR, pp. 107–120 (2001)
12. Marek, W., Nerode, A., Remmel, J.: How Complicated is the Set of Stable Models of a Recursive Logic Program? Annals of Pure and Applied Logic 56, 119–135 (1992)

13. Minker, J. (ed.): Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann, San Francisco (1988)
14. Motik, B., Horrocks, I., Sattler, U.: Bridging the Gap Between OWL and Relational Databases. In: Proc. of WWW 2007, pp. 807–816 (2007)
15. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence 138, 181–234 (2002)
16. Woltran, S.: Answer Set Programming: Model Applications and Proofs-of-Concept. Technical Report WP5, Working Group on Answer Set Programming (WASP, IST-FET-2001-37004) (July 2005), available at
`http://www.kr.tuwien.ac.at/projects/WASP/report.html`