

FDNC: Decidable Nonmonotonic Disjunctive Logic Programs with Function Symbols

THOMAS EITER and MANTAS ŠIMKUS

Technische Universität Wien

We present the class **FDNC** of logic programs which allows for function symbols (**F**), disjunction (**D**), non-monotonic negation under the answer set semantics (**N**), and constraints (**C**), while still retaining the decidability of the standard reasoning tasks. Thanks to these features, **FDNC** programs are a powerful formalism for rule-based modeling of applications with potentially infinite processes and objects, and which allows also for common-sense reasoning in this context. This is evidenced, for instance, by tasks in reasoning about actions and planning: brave and open queries over **FDNC** programs capture the well-known problems of plan existence and secure (conformant) plan existence, respectively, in transition-based actions domains. As for reasoning from **FDNC** programs, we show that consistency checking and brave/cautious reasoning tasks are **EXPTIME**-complete in general, but have lower complexity under syntactic restrictions that give rise to a family of program classes. Furthermore, we also determine the complexity of open queries (i.e., with answer variables), for which deciding non-empty answers is shown to be **EXSPACE**-complete under cautious entailment. Furthermore, we present algorithms for all reasoning tasks that are worst-case optimal. The majority of them resorts to a finite representation of the stable models of an **FDNC** program that employs maximal founded sets of knots, which are labeled trees of depth at most 1 from which each stable model can be reconstructed. Due to this property, reasoning over **FDNC** programs can in many cases be reduced to reasoning from knots. Once the knot-representation for a program is derived (which can be done off-line), several reasoning tasks are not more expensive than in the function-free case, and some are even feasible in polynomial time. This knowledge compilation technique paves the way to potentially more efficient online reasoning methods not only for **FDNC**, but also for other formalisms.

Categories and Subject Descriptors: I.2.3 [**Deduction and Theorem Proving**]: Answer/reason extraction, Inference engines, Logic programming, Nonmonotonic reasoning and belief revision; I.2.4 [**Knowledge Representation Formalisms and Methods**]: Predicate logic, Representation languages; F.4.1 [**Mathematical Logic**]: Computational logic

General Terms: Languages, Theory

Additional Key Words and Phrases: answer set programming, nonmonotonic logic programs, function symbols, computational complexity, knowledge compilation, reasoning about actions, description logics

Some of the results in this paper have been presented, in preliminary form, at the 14th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2007), Yerevan, Armenia, October 15-19, 2007.

Authors' addresses: T. Eiter, M. Šimkus, Institut für Informationssysteme, Technische Universität Wien, Favoritenstraße 9-11, A-1040 Vienna, Austria. Email: (eiter|simkus)@kr.tuwien.ac.at

This work was partially funded by the Austrian Science Fund (FWF) projects P17212 and P21840, and the EC project REWERSE (IST-2003-506779).

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 2009 ACM 1529-3785/2009/0700-0001 \$5.00

1. INTRODUCTION

Answer Set Programming (ASP) is a declarative problem solving paradigm that emerged from Logic Programming and Non-Monotonic Reasoning [Baral 2002; Lifschitz 2002; Marek and Truszczyński 1999; Niemelä 1999], and is particularly well-suited for modeling and solving problems that involve common-sense reasoning. It is based on non-monotonic logic programs under the Answer Set Semantics (also known as stable model semantics) [Gelfond and Lifschitz 1991], which assigns a given program one, no, or multiple answer sets; this facilitates to encode many problems into logic programs such that their solutions correspond to the answer sets of the programs and can be easily extracted from them. This paradigm has been successfully applied to a range of applications including data integration, configuration, reasoning about actions and change, etc. We refer the reader to [Woltran 2005] for a more detailed discussion and an overview of applications, whose number has rapidly increased in the last years.

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 [Eiter et al. 1997], Smodels [Niemelä and Simons 1997], clasp [Gebser et al. 2007] and other efficient solvers (see [Asparagus 2005]) are based in essence 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. [Bonatti 2004]. As one is forced to work with finite domains, potentially infinite processes cannot be represented naturally in ASP; additional tools to simulate unbounded domains must be used. A notable example is the DLV^K front-end of DLV [Eiter et al. 2003] which implements the action language \mathcal{K} [Eiter et al. 2004]. Constants are used to instantiate a sufficiently large domain (estimated by the user) for solving the problem; this may incur high space requirements, and does not scale to large instances.

Function symbols, in turn, are a very convenient means for generating infinite domains and objects, and allow a more natural representation of problems in such domains. However, they have been banned in ASP for a good reason: allowing them leads to undecidability even for (rather simple) Horn programs [Andreka and Nemeti 1978],¹ and negation under the answer set semantics, leads to high undecidability, cf. [Marek et al. 1994; Marek and Remmel 2003; Marek et al. 1992]. 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. Several works have addressed this issue, including [Chomicki and Imielinski 1993; Chomicki 1995; Bonatti 2004; Baselice et al. 2007; Syrjänen 2001]. More recently, Bonatti and his co-workers introduced *finitary* and *finitely recursive logic programs* [Bonatti 2004; Baselice et al. 2007]. They imposed syntactic conditions on the groundings of logic programs, which are infinite in the presence of function symbols. Therefore, the verification of the conditions is difficult (in fact, undecidable), which limits the applicability of the results. Syrjänen [2001] used a generalization of stratification which can be effectively checked, while Chomicki and Imieliński [1993; 1995] introduced programs without negation in

¹See [Itai and Makowsky 1987] for an interesting historic account (quoted in [Dantsin et al. 2001]).

which restrictions applied to the rules individually. We refer to Section 8 for more details and discussion of related work.

In this paper, we pursue an approach to obtain decidable logic programs with function symbols by merely constraining, similarly as in [Chomicki and Imielinski 1993; Chomicki 1995], the rule 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 because they have to deal with possibly infinite models. However, because of the peculiarities of Answer Set Semantics, transferring these results to logic programs with function symbols is not straightforward. Reasoning with logic programs needs to be more refined since only Herbrand models of a program are of interest and, moreover, only particular such models (fulfilling the condition of stability), which happen to be special minimal Herbrand models.

The main contributions of this paper are briefly summarized as follows.

- We introduce the class \mathbb{FDNC} of logic programs, which allow function symbols (\mathbb{F}), disjunction (\mathbb{D}), non-monotonic negation (\mathbb{N}) under the answer set semantics [Gelfond and Lifschitz 1991], and constraints (\mathbb{C}). In order to provide decidable reasoning, \mathbb{FDNC} programs are syntactically restricted to ensure that they have the *forest-shaped model* property. To this end, in the first stage programs are considered in which the predicates are unary and binary, and function symbols are unary; this gives us the class of ordinary \mathbb{FDNC} programs, described in Section 3. To accommodate predicates of higher arity, an extension of \mathbb{FDNC} to higher-arity predicates is conceived in Section 7. The syntactic restrictions are similar to those in [Chomicki and Imielinski 1993] and limit the use of functions symbols, but are more restrictive. However, they enable us to develop special techniques for handling \mathbb{FDNC} -programs, which are needed in order to cope with negation, disjunction, and constraints, which were not considered in [Chomicki and Imielinski 1993]. Furthermore, we consider the natural restrictions of \mathbb{FDNC} programs that arise if the constructs of negation (\mathbb{N}), disjunction (\mathbb{D}) and constraints (\mathbb{C}) are disallowed, giving rise to a whole family of logic programs ranging from \mathbb{F} to \mathbb{FDNC} ; the plainest language in this family, \mathbb{F} , is a subclass of Horn programs that is (apart from minor deviations) a fragment of *Datalog_{NS}* in [Chomicki and Imielinski 1993].
- We study standard reasoning tasks for \mathbb{FDNC} , including deciding the consistency of a program P , i.e., existence of an answer set of P , as well as brave and cautious entailment of a ground atom A from a program P . Furthermore, we also consider brave and cautious entailment of existential queries of the form $\exists \vec{x}.Q(\vec{x})$, where Q is a predicate and \vec{x} is a tuple of variables, and also of open queries $\lambda \vec{x}.Q(\vec{x})$ for which the variables \vec{x} must be bound to ground terms prior to entailment checking. For these problems, we develop algorithms and characterize their computational complexity over the whole program family from \mathbb{F} to \mathbb{FDNC} , in terms of completeness results for suitable complexity classes. As we show, for \mathbb{FDNC} all reasoning tasks are EXPTIME-complete, with the exception of deciding answer existence for open queries under cautious entailment, which is EXPSPACE-complete. Disallow-

ing either disjunction and constraints (which gives $\mathbb{F}\mathbb{N}$) or non-monotonic negation (which gives $\mathbb{F}\mathbb{D}\mathbb{C}$) does not lead to lower complexity, while all problems drop to PSPACE-completeness if both negation and disjunction are disallowed (which gives $\mathbb{F}\mathbb{C}$, that are Horn logic programs with constraints). Depending on the reasoning task and the constructs available, the complexity ranges in the other cases from polynomial time over co-NP , Σ_2^P , PSPACE and EXPTIME up to EXPSPACE. In particular, for \mathbb{F} programs (which are Horn programs), entailment of ground atoms is polynomial; note that even in the absence of function symbols, this problem is NP-hard for Horn programs with binary predicates. Table I compactly summarizes the complexity results in Section 4, which also provides a detailed discussion.

- The EXPTIME-hardness proofs for consistency checking of programs in the fragments $\mathbb{F}\mathbb{N}$, $\mathbb{F}\mathbb{D}\mathbb{C}$, and $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$, are by a reduction from satisfiability testing in the EXPTIME-complete Description Logic \mathcal{ALC} . As a side result, we obtain a polynomial time mapping of a well-known Description Logic (cf. [Baader et al. 2003]) to logic programs under answer set semantics. The mapping takes advantage of a normal form of \mathcal{ALC} knowledge bases and is balanced in the sense that it maps to a class of logic program whose complexity is not higher than the one of \mathcal{ALC} (see Section 8 for a discussion of other mappings). These results are interesting in their own right and may be exploited in other contexts.
- $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ programs can have infinite and infinitely many stable models, which therefore can not be explicitly represented for reasoning purposes. We provide a method to finitely represent all the stable models of a given $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ 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 at most 1. The finite representation technique allows us to define an elegant decision procedure for brave reasoning in $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$. It may also be exploited for offline *knowledge compilation* [Cadoli and Donini 1997; Darwiche and Marquis 2002] to speed up online reasoning, by precomputing and storing a knot representation of a logic program P . Given such a representation, multiple query answering from P can be done comparatively efficiently (some problems are solvable in polynomial time), and also model building can be supported (which is of concern in ASP): with the knots as building blocks, any stable model of P can be gradually constructed (leading to an infinite process in general), at no higher cost than in the function-free case. In general, a knot representation of a logic program is exponential in the program size; this is the common tradeoff between time and space for such compilation, and is encountered in other compilation forms as well (e.g., compilation of a propositional formula into all its prime implicates [Darwiche and Marquis 2002]).

Thanks to their features, $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ programs are a powerful formalism for rule-based modeling of applications with potentially infinite processes or objects, which also accommodates common-sense reasoning through non-monotonic negation. From a complexity perspective, $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ and its subclasses provide *effective syntax* for expressing problems in PSPACE, EXPTIME and EXPSPACE using logic programs with function symbols.

This can, for instance, be fruitfully exploited for reasoning about actions and planning. The usability of answer set programming in this area is well-known and has been explored in many works, including [Dimopoulos et al. 1997; Lifschitz 1999;

Baral 2002; Eiter et al. 2004; Tu et al. 2007; Son et al. 2006; Son et al. 2005; Morales et al. 2007]; the excellent book [Baral 2002] (recommended for background) devotes a whole chapter to this subject. FDNC programs allow to encode action domain descriptions in transition-based action formalisms which support incomplete states and nondeterministic action effects, like \mathcal{C} [Giunchiglia and Lifschitz 1998], \mathcal{K} [Eiter et al. 2004], or fragments of the situation calculus (see e.g. [Levesque et al. 1998] for background) such that arbitrarily long action sequences can be naturally handled.

As an appetizer for the use of FDNC programs in this area, we sketch here informally elements of a simple encoding of a plain propositional variant of the situation calculus into FDNC programs. To this end, we use unary predicates $F(x)$ for fluents F that describe the state of the domain in a certain situation, a unary predicate $S(x)$ to denote situations x , and the constant *init* for the initial situation. For the latter, a fact $S(\text{init}) \leftarrow$ is set up, and the initial state of the domain is then described by facts of the form $F(\text{init}) \leftarrow$.

Transitions happen through the execution of actions A_1, \dots, A_n , which are represented by function symbols f_{A_1}, \dots, f_{A_n} ; intuitively, $f_{A_i}(x)$ is the situation resulting if action A_i is taken in situation x . Using a binary predicate Tr , we can express that a transition happened by $Tr(x, f_{A_i}(x))$; a rule $A_1(x) \vee \dots \vee A_n(x) \leftarrow S(x)$ singles out some action in situation x for moving on. If the action A_i can be taken, which is assessed by some predicate $Poss_{A_i}(x)$, then the transition is made, described by the rule $Tr(x, f_{A_i}(x)) \leftarrow A_i(x), Poss_{A_i}(x)$; the new situation after taking an action is described with $S(y) \leftarrow Tr(x, y)$.

These rules and facts provide a generic backbone for describing an evolving action domain. Particular action effects during transitions can be stated by rules of FDNC ; e.g., the rule $F(f_\alpha(x)) \leftarrow Tr(x, f_\alpha(x))$ states that after executing the action α , F holds in the follow up situation. Importantly, the availability of non-monotonic negation allows to conveniently state fluent inertia, i.e., the fluent value when taking an action remains the same *by default*. For fluent F , this can be expressed using the two rules

$$\begin{aligned} F(y) &\leftarrow F(x), Tr(x, y), \text{not } \bar{F}(y), \\ \bar{F}(y) &\leftarrow \bar{F}(x), Tr(x, y), \text{not } F(y), \end{aligned}$$

where $\bar{F}(x)$ is a predicate for the complement of $F(x)$ that can be simulated by adding the constraint $\leftarrow F(x), \bar{F}(x)$. Possible states of the domain in a situation (in case of incomplete information) can be captured by rules $F(x) \vee \bar{F}(x) \leftarrow S(x)$. Overall, the stable models of the program will then correspond to trajectories of the action domain, i.e., sequences of actions together with the fluent values at each stage of action execution. If we replace the disjunctive rule $A_1(x) \vee \dots \vee A_n(x) \leftarrow S(x)$ with the rules $A_1(x) \leftarrow S(x); \dots; A_n(x) \leftarrow S(x)$, then the stable models correspond to the unwindings of the initial state according to the possible transitions.

Using these elements, FDNC may be used to represent a number of action domains from the literature, e.g., the Yale Shooting [Hanks and McDermott 1987], Bomb in the Toilet, and others cf. [Eiter et al. 2004], and to solve reasoning and planning problems on them. In Appendix B we more concretely elaborate on an encoding of action domains in a fragment of the language \mathcal{K} into FDNC , and show in an example how query answering can be used to elegantly solve, among others, conformant planning problems in \mathcal{K} . The latter are EXPSpace -complete in general,

and show that $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ -programs offer the complexity tailored to these problems.

The remainder of this paper is organized as follows. Section 2 briefly introduces the basic concepts and notation of disjunctive logic programs used in this paper. Section 3 then introduces $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ programs, and establishes their basic semantic properties. It also introduces the finite representation of stable models in terms of knots. Section 4 gives an overview and a discussion of the complexity results in this paper, which are established in the subsequent Sections 5 and 6. In the course of this, also reasoning techniques and algorithms are developed. Section 7 considers an extension of ordinary to higher-arity $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ programs. The final Sections 8 and 9 discuss related work and conclude with issues for future work. Details of some proofs and constructions can be found in the appendix.

2. PRELIMINARIES

We assume fixed countably infinite sets of *constant symbols*, *function symbols*, *predicate symbols*, and *variables*. Moreover, each function and relation symbol has an associated positive integer, its *arity*. A *term* is either a constant symbol, a variable or an expression of the form $f(\vec{t})$ where f is a function symbol, \vec{t} is an n -tuple of terms and n is the arity of f . An *atom* is an expression of the form $R(\vec{t})$ where R is a predicate symbol, \vec{t} is an n -tuple of terms and n is the arity of R . An atom is also called a *positive literal*. An expression of the form *not* A , where A is an atom, is a *negative literal*. A *literal* is either a positive or a negative literal.

A *disjunctive logic program* (or *program*) is a set of (*disjunctive*) *rules* r of form

$$A_1 \vee \dots \vee A_n \leftarrow L_1, \dots, L_m, \quad (1)$$

where $n + m > 0$, A_1, \dots, A_n are atoms and L_1, \dots, L_m are literals. The atoms A_1, \dots, A_n are the *head atoms* of r and L_1, \dots, L_m are the *body literals* of r . We let $\text{head}(r) = \{A_1, \dots, A_n\}$ and denote by $\text{body}^+(r)$ (resp., $\text{body}^-(r)$) the set of atoms that occur in positive (resp., negative) body literals of r . A (*disjunctive*) *fact* is a rule (1) with empty body ($m = 0$), also written $A_1 \vee \dots \vee A_n$, while a *constraint* is a rule with no atoms in the head ($n = 0$). If $\text{body}^-(r) = \emptyset$, then r is *positive*, and we let $\text{body}(r) = \text{body}^+(r)$. A program is *positive*, if it contains only positive rules.

The semantics of a program P is given in terms of Herbrand interpretations. Let \mathcal{HU}^P be the *Herbrand universe* of P , i.e., the set of all terms that can be built from constants and function symbols occurring in P . Similarly, \mathcal{HB}^P is the *Herbrand base* of P , i.e., the set of all atoms that can be built from predicate symbols of P and terms in \mathcal{HU}^P . An (*Herbrand*) *interpretation* for P is an arbitrary subset of \mathcal{HB}^P .

A term, atom, rule, program is *ground*, if it contains no variables. A rule r' is a *ground instance* of a rule $r \in P$, if r' is a ground rule obtained from r by replacing each variable in r by a term in \mathcal{HU}^P ; by $\text{Ground}(P)$ we denote the set of all ground instances of the rules in P .

A (Herbrand) interpretation I *satisfies* a ground rule r , denoted $I \models r$, if $\text{body}^+(r) \subseteq I \wedge \text{body}^-(r) \cap I = \emptyset$ implies $I \cap \text{head}(r) \neq \emptyset$. An interpretation I is a *model* of a program P , denoted $I \models P$, if I satisfies each rule $r \in \text{Ground}(P)$; moreover, I is a *minimal model* of P , if no $J \subset I$ is a model of P . The set of minimal models of P is denoted by $\text{MM}(P)$.

Given an interpretation I for a program P , the *Gelfond-Lifschitz reduct* [Gelfond and Lifschitz 1991] of P , denoted P^I , is obtained from $\text{Ground}(P)$ by

- (i) removing all rules r such that $\text{body}^-(r) \cap I \neq \emptyset$, and
- (ii) removing all negative literals from the remaining rules.

Then I is a *stable model* (or *answer set*) of P , if $I \in \text{MM}(P^I)$. The set of all stable models of P is denoted by $\text{SM}(P)$. A program P is *consistent*, if $\text{SM}(P) \neq \emptyset$.

A *ground (atomic) query* is a ground atom A , and an *existential (atomic) query* is an expression $\exists \vec{x}.Q(\vec{x})$, where \vec{x} is an n -tuple of variables and Q is an n -ary predicate symbol. An *open query* is a similar expression $\lambda \vec{x}.Q(\vec{x})$.

As usual, a program P *bravely entails* a ground query A (resp., an existential query $\exists \vec{x}.Q(\vec{x})$), denoted $P \models_b A$ (resp., $P \models_b \exists \vec{x}.Q(\vec{x})$), if A (resp. $\exists \vec{x}.Q(\vec{x})$) is true in some stable model I of P , i.e., I contains A (resp., some ground atom $Q(\vec{t})$). Moreover, P bravely entails an open query $\lambda \vec{x}.Q(\vec{x})$, denoted $P \models_b \lambda \vec{x}.Q(\vec{x})$, if P bravely entails some ground query $Q(\vec{t})$; any such \vec{t} is called an *answer* for the query.

The notion of cautious entailment, \models_c , is dually defined, where “every stable model” replaces “some stable model.” Note that $P \models_b \lambda \vec{x}.Q(\vec{x})$ iff $P \models_b \exists \vec{x}.Q(\vec{x})$, while $P \models_c \lambda \vec{x}.Q(\vec{x})$ implies $P \models_c \exists \vec{x}.Q(\vec{x})$ but not vice versa; this is because $\lambda \vec{x}$ requires that \vec{t} is the *same* in all stable models, while $\exists \vec{x}$ permits varying terms in different stable models. Cautious entailment of open queries is a useful tool e.g. in planning, to determine *conformant (alias secure) plans*, i.e., sequences of actions whose execution leads to the goal, regardless of possibly incomplete knowledge about the initial state and/or nondeterministic action effects (see Appendix).

EXAMPLE 2.1. Consider the program P consisting of the following rules:

$$\begin{array}{l} D(a) \leftarrow \\ B(f(x)) \leftarrow D(x), \text{not } A(x) \qquad C(x) \leftarrow A(x) \\ A(x) \leftarrow D(x), \text{not } B(f(x)) \qquad C(x) \leftarrow B(x) \end{array}$$

P has two stable models $I_1 = \{D(a), B(f(a)), C(f(a))\}$ and $I_2 = \{D(a), A(a), C(a)\}$. This is because I_1 and I_2 are the minimal models of P^{I_1} and P^{I_2} , respectively, where

$$\begin{array}{l} P^{I_1} = \{D(a) \leftarrow, \\ B(f^{i+1}(a)) \leftarrow D(f^i(a)), \\ A(f^{i+1}(a)) \leftarrow D(f^{i+1}(a)), \\ C(f^i(a)) \leftarrow A(f^i(a)), \\ C(f^i(a)) \leftarrow B(f^i(a)) \mid i \geq 0\} \\ P^{I_2} = \{D(a) \leftarrow, \\ B(f^{i+2}(a)) \leftarrow D(f^{i+1}(a)), \\ A(f^i(a)) \leftarrow D(f^i(a)), \\ C(f^i(x)) \leftarrow A(f^i(x)), \\ C(f^i(x)) \leftarrow B(f^i(x)) \mid i \geq 0\} \end{array}$$

No other interpretation is a stable model of P . Note that $P \models_b \exists x.B(x)$ and $P \models_c \exists x.C(x)$, while $P \not\models_c \lambda x.C(x)$, i.e., $\lambda x.C(x)$ has no answer. On the other hand, $P \models_c \lambda x.D(x)$ and $x = a$ is an answer of $\lambda x.D(x)$.

3. FDNC PROGRAMS

We now introduce the class FDNC of logic programs with function symbols. The syntactic restrictions that are imposed ensure the decidability of the formalism, but allow infinitely many and possibly infinite stable models. We then analyze the model-theoretic properties of FDNC programs and introduce a method to finitely represent the (possibly infinite) collection of stable models of a program. For convenience, we use $P^\pm(\vec{t})$ to generically denote one of the literals $P(\vec{t})$ and $\text{not } P(\vec{t})$.

DEFINITION 3.1 (FDNC PROGRAMS). An FDNC program is a finite disjunctive logic program whose rules are of the following forms:

$$\begin{aligned}
(R1) \quad & A_1(x) \vee \dots \vee A_k(x) \leftarrow B_0(x), B_1^\pm(x), \dots, B_l^\pm(x) \\
(R2) \quad & R_1(x, y) \vee \dots \vee R_k(x, y) \leftarrow P_0(x, y), P_1^\pm(x, y), \dots, P_l^\pm(x, y) \\
(R3) \quad & R_1(x, f_1(x)) \vee \dots \vee R_k(x, f_k(x)) \leftarrow P_0(x, g_0(x)), P_1^\pm(x, g_1(x)), \dots, P_l^\pm(x, g_l(x)) \\
(R4) \quad & A_1(y) \vee \dots \vee A_k(y) \leftarrow R_0(x, y), R_1^\pm(x, y), \dots, R_l^\pm(x, y), \\
& \quad B_1^\pm(x), \dots, B_m^\pm(x), C_1^\pm(y), \dots, C_n^\pm(y) \\
(R5) \quad & A_1(f(x)) \vee \dots \vee A_k(f(x)) \leftarrow R_0(x, f(x)), R_1^\pm(x, f(x)), \dots, R_l^\pm(x, f(x)), \\
& \quad B_1^\pm(x), \dots, B_m^\pm(x), C_1^\pm(f(x)), \dots, C_n^\pm(f(x)) \\
(R6) \quad & R_1(x, f_1(x)) \vee \dots \vee R_k(x, f_k(x)) \leftarrow B_0(x), B_1^\pm(x), \dots, B_l^\pm(x) \\
(R7) \quad & C_1(\vec{c}_1) \vee \dots \vee C_k(\vec{c}_k) \leftarrow D_1^\pm(\vec{b}_1), \dots, D_l^\pm(\vec{b}_l),
\end{aligned}$$

where $k, l, m, n \geq 0$, and each \vec{c}_i, \vec{b}_i is a tuple of constants of arity ≤ 2 . Moreover, at least one rule in the program is of type (R7). W.l.o.g., we assume that in one-variable (resp., two-variable) rules, the variable in unary atoms (resp., variable tuple in binary atoms) is always x (resp., $\langle x, y \rangle$).

The fragments obtained from FDNC by disallowing disjunction, constraints or negative literals are denoted by omitting respectively \mathbb{D} , \mathbb{C} , and \mathbb{N} in the name. The collection of all these fragments is called the \mathbb{F} family.

The restrictions draw their inspiration from classical first-order clauses with existential quantification restricted to positive literals, i.e., of implications $\forall \vec{x} \exists \vec{y} \alpha(\vec{x}) \rightarrow \beta(\vec{x}, \vec{y})$ where $\alpha(\vec{x})$ is a conjunction and $\beta(\vec{x}, \vec{y})$ is a disjunction of atoms with free variables \vec{x} and \vec{x}, \vec{y} , respectively. Of particular interest are clauses with predicate arities ≤ 2 where existential quantification is additionally restricted to one variable in binary literals. Skolemization eliminates each existentially quantified variable with a fresh unary function symbol; the Herbrand universe of a theory can then be represented by a labeled graph that has certain tree shape: terms $f(t)$ being children of a term t . The rules allow to describe unary predicates satisfied by terms (classification), and to define binary relationships between them. In particular, we can describe properties of a term t depending solely on t itself (by rules (R1)), and relations between t and another term t' depending on other existing relations (by rules (R2)). By rules (R4), we can talk about how the properties of a term t affect the properties of terms to which t is related. Rules (R6) are crucial as they allow to introduce new objects: we can state the existence of a child term $f(t)$ to which t is related. Such use of function symbols is convenient in applications, and ensures the forest-model property on which decidability and complexity proofs hinge. With rules (R3), we can describe further relations between t and a term $f(t)$ depending on other such relationships for terms $g(t)$, and with rules (R5) properties of $f(t)$, depending on relations between t and $f(t)$ and properties of t and other properties of $f(t)$. Finally, the rules (R7) allow us to express arbitrary properties of and binary relations between elementary objects (represented by constants).

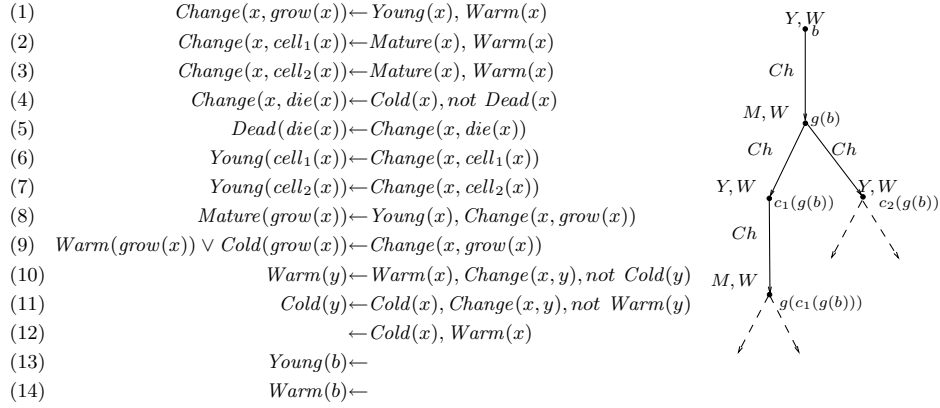


Fig. 1. Example: Evolution of a Cell

We note that the rules (R5) are (non-ground) instances of (R4), and thus not strictly needed; in turn, rules (R4) can be eliminated using rules (R5) and (R7). Similarly, the rules (R2) could be equivalently replaced by rules (R3) and (R7). However, (R2) and (R5) are useful for modeling purposes and thus included.

The first body atom in the rules (R1)-(R6) ensures their safety, i.e., each variable occurs in a positive body atom. For (R1), (R3), (R5) and (R6) this could be relaxed (no positive body atom is prescribed). Such non-safe programs can be simulated by $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ programs using a unary *domain predicate* and a binary *successor predicate* that holds for each term t and each pair $\langle t, f(t) \rangle$ of terms, respectively, in the Herbrand universe.² By eliminating rules (R2) and (R4) beforehand, we could have a variant of $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ without safety restrictions; the connected forest-shaped models of $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ programs would change into a rudimentary form.

The structure of the rules in $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ syntax, the availability of non-monotonic negation and function symbols allows us to represent possibly infinite processes in a rather natural way. We provide here an example from the biology domain.

EXAMPLE 3.2. *As a running example, we use the $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ program P in Figure 1. It represents the evolution of a cell, viz. growth, splitting into two cells, and death. (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)-(8) determine whether a cell is dead, young or mature. The rules (9)-(11) state the knowledge about the temperature. During growth (which takes some time), it might alter, while in the other changes (which happen quickly), it stays the same, which is expressed by inertia rules (10) and (11). Finally, (13) and (14) are the initialization facts. (For brevity, we also shorten predicate symbols to W (arm), C (old), Y (oung), M (ature), D (ead), and Ch (ange) and function symbols to c (ell)₁, c (ell)₂, g (row), d (ie).)*

²Using fresh unary and binary predicates Dom and $Succ$, respectively, augment P with (i) $Dom(c) \leftarrow$ for each constant c of P , (ii) $Succ(x, f(x)) \leftarrow Dom(x)$ for each function symbol f of P , (iii) and the rule $Dom(y) \leftarrow Succ(x, y)$. Finally, add in the body of each original rule r the atom $Dom(x)$ if r is of form (R1), (R3), or (R6), and $Succ(x, f(x))$ if r is form (R5). As easily seen, the rewriting preserves stable models on the initial signature.

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 part of the stable model that is depicted in Figure 1 represents a development where the temperature does not change during the growth of b and its child. Another stable model is $I = \{ \text{Young}(b), \text{Warm}(b), \text{Change}(b, \text{grow}(b)), \text{Cold}(\text{grow}(b)), \text{Mature}(\text{grow}(b)), \text{Change}(\text{grow}(b), \text{die}(\text{grow}(b))), \text{Dead}(\text{die}(\text{grow}(b))), \text{Cold}(\text{die}(\text{grow}(b))) \}$ which corresponds to the situation that the temperature changes and the bacterium dies.

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

Example 3.2 shows that in presence of function symbols, an FDNC program may have infinite stable models. We note that FDNC programs do not have the finite-model property, i.e., a program might have only infinite stable models. This is witnessed by the simple F program $P = \{A(c) \leftarrow ; R(x, f(x)) \leftarrow A(x); A(y) \leftarrow R(x, y)\}$, whose single stable model contains infinitely many atoms.

Due to the lack of finite-model property, the search for stable models of an FDNC program P cannot be confined to a finite search-space, i.e., consistency cannot be decided by considering a finite subset of the grounding of the program. 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 P .

3.1 Characterization of Stable Models

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

DEFINITION 3.3. *An (Herbrand) interpretation I is forest-shaped, if the following hold:*

- (a) *All the atoms in I are either unary or binary. Additionally, each binary atom in I is of the form $R(c, d)$ or $R(t, f(t))$, where c, d are constants, and t is a ground 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 , $R(t, f(t)) \in I$.*

The “graph part” of I consists of the atoms $R(c, d)$, where c, d are constant symbols; intuitively, c and d are connected by an arc from c to d . The other binary atoms constitute a set of trees, as $f(t)$ has via $R(t, f(t))$ the term t as its uniquely determined ancestor, and the root of each such tree must be a constant symbol (i.e., a node of the graph part). The following proposition is important.

PROPOSITION 3.4. *If H is an arbitrary interpretation of an FDNC program P and $J \in \text{MM}(P^H)$, then J is forest-shaped (in particular, every $J \in \text{SM}(P)$).*

PROOF. The property follows directly from the structure of the rules and the minimality requirements. In particular, for (a) in Definition 3.3, note that the rules of P can have binary atoms only of the forms $R(c, d)$, $R(x, f(x))$ and $R(x, y)$. In the

case of $R(x, y)$ atoms in the head (case of (R2) rules), the body atom arguments are $\langle x, y \rangle$, and hence such rules do not spoil the argument structure, i.e., they cannot introduce atoms of a shape different from the one in (a). For (b), note that the atoms of the form $A(f(t))$ can be derived only via the rules (R1), (R4) or (R5). Firing rules (R4) and (R5) requires an atom of the form $R(t, f(t))$. In the case of rules (R1), all body atoms have the same term as in the head and hence the derivation of $A(f(t))$ can be traced back to rules (R4) or (R5). Suppose H is an arbitrary interpretation for P . Assume some $J \in MM(P^H)$ contains an atom violating (a) or (b) in Definition 3.3. We can simply collect all the atoms violating (a) or (b) and remove them from J . Due to the observations above, such removal does not violate any rule in P^H , and, hence, we have that J is not minimal. Contradiction. The second claim follows from the definition of stable models. \square

The methods that we present in this paper are aimed at providing the decidability results together with the worst-case optimal algorithms for $\mathbb{F}DNC$. We note, however, that the decidability of the reasoning tasks discussed in this paper can be inferred from the results in [Eiter and Gottlob 1997]. The technique in [Eiter and Gottlob 1997] shows how the stable model semantics for 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 $\mathbb{F}DNC$ programs in monadic second-order logic over trees SkS , which is known to be decidable (see [Motik et al. 2007] for a related encoding). Unfortunately, optimal algorithms or exact complexity characterizations are not apparent from such encodings, which are usually processed using automata-based algorithms.

The semantic characterization and the reasoning methods later on follow an intuition that stable models for an $\mathbb{F}DNC$ 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 $\text{Ground}(P)$ and adding facts (*states*) obtained in the previous iteration.

In the rest of Section 3, we assume that P is an arbitrary $\mathbb{F}DNC$ program. For convenience, given a term t and a set of atoms I , we write $t \hat{\in} I$, if there exists an atom in I having t as an argument. We next define states and atomic state sets associated with sets of atoms and programs.

DEFINITION 3.5 (STATES $\text{st}(I, t)$; ATOMIC STATE SETS $\text{st}(I), \text{st}(P)$). *A state of any ground term t is an arbitrary set U^t of unary atoms of form $A(t)$. For any set of atoms I and term $t \hat{\in} I$, the state of t in I is $\text{st}(I, t) = \{A(t) \mid A(t) \in I\}$. Furthermore, the atomic state set of I (resp., a program P) is $\text{st}(I) = \{\text{st}(I, c) \mid c \hat{\in} I \text{ is a constant}\}$ (resp., $\text{st}(P) = \bigcup_{I \in SM(P)} \text{st}(I)$).*

EXAMPLE 3.6 (CONT'D). *For the above stable model I of P , we have $\text{st}(I, b) = \{\text{Young}(b), \text{Warm}(b)\}$, $\text{st}(I, \text{grow}(b)) = \{\text{Cold}(\text{grow}(b)), \text{Mature}(\text{grow}(b))\}$, and $\text{st}(I, \text{die}(\text{grow}(b))) = \{\text{Dead}(\text{die}(\text{grow}(b))), \text{Cold}(\text{die}(\text{grow}(b)))\}$. Moreover, $\text{st}(I) = \{\text{st}(I, b)\}$, and as all stable models of P clearly agree on the function-free atoms, $\text{st}(P) = \text{st}(I)$.*

We omit t from U^t if t is not of particular interest. For a one-variable rule r in $\mathbb{F}DNC$ syntax and a term t , let $r_{\downarrow t}$ denote the rule obtained by substituting the

| K_1 | K_2 | K_3 |
|--|--|--|
| $M(g(b)), W(g(b)),$ $Ch(g(b), c_1(g(b))),$ $Y(c_1(g(b))), W(c_1(g(b))),$ $Ch(g(b), c_2(g(b))),$ $Y(c_2(g(b))), W(c_2(g(b)))$ | $M(g(b)), Y(g(b)), W(g(b))$ $Ch(g(b), c_1(g(b))),$ $Ch(g(b), c_2(g(b))),$ $Ch(g(b), g(g(b))),$ $Y(c_1(g(b))), W(c_1(g(b))),$ $Y(c_2(g(b))), W(c_2(g(b))),$ $M(g(g(b))), C(g(g(b))),$ | $Y(b), W(b),$ $Ch(b, g(b)), M(g(b)),$ $Y(g(b)), W(g(b))$ |
| | | |

Fig. 2. Example knots

variable x in r with t . Similarly, for a two-variable r and terms s, t , let $r_{\downarrow s, t}$ denote the rule obtained by substituting x and y in r with s and t , respectively.

DEFINITION 3.7 (LOCAL PROGRAM $P(U^t)$). *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., $U = \text{st}(I, t)$. 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 induce a stable model of $P(U)$. Stable models of local programs have a simple structural property, captured by the notion of *knots*.

DEFINITION 3.8 (KNOT). *A knot with 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).*

We say K 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). Let $\text{succ}(K)$ denote the set of all terms $f(t) \hat{\in} K$.

A knot with root term t can be viewed as a labeled tree of depth at most 1, where $\text{succ}(K)$ are the leaves. 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. Figure 2 shows an example of knots over the signature of the program P in Example 3.2.

It is easy to see that due to the structure of local programs, their stable models satisfy the conditions in Definition 3.8 and hence are knots. On the other hand, knots are also the structures that occur in the trees of the forest-shaped interpretations. To “extract” knots from such interpretations, the following is 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 for P and $t \in I$, the set $K = I \cap \mathcal{HB}_t$ is a knot over P .

The following notion of *stable knot* is central. Stable knots are self-contained building blocks for stable models of FDNC programs.

DEFINITION 3.9 (STABLE KNOT). *Let K be a knot with root term t and $U^t = \text{st}(K, t)$. 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 root term t and $U^t = \text{st}(K, t)$ is stable w.r.t. P , and that t occurs in a forest-shaped interpretation I for P as a “leaf node”, i.e., I has no atoms of form $R(t, f(t))$. If the states of t in I and K coincide, i.e., $\text{st}(I, t) = U^t$, then intuitively K is a suitable set of atoms to give t the necessary successors in I .

EXAMPLE 3.10 (CONT'D). *Consider the knots K_1, K_2 and K_3 in Figure 2. As easily seen, P has a stable model I in which K_1 occurs, i.e., $I \cap \mathcal{HB}_{g(b)} = K_1$; in fact, Figure 1 shows an example. In contrast, K_2 and K_3 do not occur in any stable model of P , as the rules of P do not force an element to satisfy both M and Y .*

The knot K_1 is stable: as easily checked, K_1 is a stable model of the local program $P(\{M(g(b)), W(g(b))\})$. While K_2 does not occur in any stable model of P , it is a stable model of $P(\{M(g(b)), Y(g(b)), W(g(b))\})$, and hence stable. Intuitively, K_2 is an eligible building block for a stable model of P only if $g(b)$ satisfies exactly W and both M and Y . The knot K_3 is not stable, since the stable models of $P(\{Y(b), W(b)\})$ are $K_3 \setminus \{Y(g(b))\}$ and $K_3 \setminus \{Y(g(b)), W(g(b))\} \cup \{C(g(b))\}$.

After introducing the necessary notions for the tree-part of forest-shaped interpretations, we turn to the graph part.

DEFINITION 3.11 (GRAPH PROGRAM $\text{gp}(P)$). *For a program P , by $\text{gp}(P)$ we denote the set of all function-free rules $r \in \text{Ground}(P)$.*

EXAMPLE 3.12 (CONT'D). *In our running example, $\text{gp}(P)$ consists of the two facts $\text{Young}(b) \leftarrow$ and $\text{Warm}(b) \leftarrow$.*

The following theorem characterizes the stable models of P . For an interpretation I , let $\text{ffa}(I)$ be the set of all function-free atoms in I .

THEOREM 3.13. *If I is an interpretation for P , then the following are equivalent:*

- (A) I is a stable model of P .
- (B) I is a forest-shaped interpretation such that (i) $\text{ffa}(I)$ is a stable model of $\text{gp}(P)$, and (ii) for each term $t \in I$, $I \cap \mathcal{HB}_t$ is a knot that is stable w.r.t. P .

PROOF. (A) \Rightarrow (B). Assume $I \in SM(P)$. By Proposition 3.4, I is forest-shaped. First, we show that (i) holds, by exploiting the concept of *modularity* in disjunctive programs under the Answer Set semantics [Eiter et al. 1997] (this is closely related to *splitting sets* of [Lifschitz and Turner 1994] for normal programs). Let $Q =$

$\text{Ground}(P) \setminus \text{gp}(P)$. Note that none of the head atoms in rules of Q occurs in rules of $\text{gp}(P)$, and hence $\text{gp}(P)$ is *independent* from Q . By Lemma 5.1 in [Eiter et al. 1997], since $I \in \text{SM}(\text{Ground}(P))$ and $\text{gp}(P)$ is independent from Q , $I \cap \mathcal{HB}^{\text{gp}(P)}$ is a stable model of $\text{gp}(P)$. Since $I \cap \mathcal{HB}^{\text{gp}(P)} = \text{ffa}(I)$, the claim holds.

We similarly show that (ii) holds. Suppose $t \hat{\in} I$ and $K = I \cap \mathcal{HB}_t$. As I is forest-shaped, K is a knot over the signature of P . Suppose K is not stable w.r.t. P , i.e., $K \notin \text{SM}(P(U))$, where $U = \text{st}(K, t)$. There are two possibilities:

- $K \not\models P(U)^K$. Then some rule $r \in P(U)^K$ exists such that $\text{body}(r) \subseteq K$ and $\text{head}(r) \cap K = \emptyset$. As each fact $A(t) \leftarrow$ is in $P(U)$ iff $A(t) \in K$, r is not of this form. Thus $r \in P_t$, where P_t results from $P(U)$ by removing the facts. By the construction of local programs, $P_t \subseteq \text{Ground}(P)$. As $K = I \cap \mathcal{HB}_t$, K and I agree on the reduct for the rules in P_t and the interpretation of their atoms. This implies $r \in P^I$, $\text{body}(r) \subseteq I$ and $\text{head}(r) \cap I = \emptyset$. Hence, $I \notin \text{SM}(P)$.
- $K \models P(U)^K$, but is not minimal, i.e., some $H \subset K$ fulfills $H \models P(U)^K$. Let $M = H \cup (I \setminus K)$. Obviously, $M \subset I$; we show that $M \models P^I$, which implies $I \notin \text{SM}(P)$. Suppose $M \not\models P^I$. Then some rule $r \in P^I$ exists such that $\text{body}(r) \subseteq M$ and $\text{head}(r) \cap M = \emptyset$. As $I \models P^I$ but $M \not\models P^I$, r has one of the following forms:
 - (a) $A_1(t) \vee \dots \vee A_k(t) \leftarrow B_0(t), \dots, B_l(t)$,
 - (b) $R_1(t, f_1(t)) \vee \dots \vee R_k(t, f_k(t)) \leftarrow P_0(t, g_0(t)), \dots, P_l(t, g_l(t))$,
 - (c) $A_1(f(t)) \vee \dots \vee A_k(f(t)) \leftarrow B_0(f(t)), \dots, B_l(f(t))$,
 - (d) $A_1(f(t)) \vee \dots \vee A_k(f(t)) \leftarrow B_1(Z_1), \dots, B_m(Z_m), R_0(t, f(t)), \dots, R_l(t, f(t))$, or
 - (e) $R_1(t, f_1(t)) \vee \dots \vee R_k(t, f_k(t)) \leftarrow B_0(t), \dots, B_l(t)$,

where each $Z_i \in \{t, f(t)\}$, and $k, l, m \geq 0$. The rules above are derived by taking all rules of P^I that have only atoms with terms t or $f(t)$ in the head. Since M results from I by removing some atoms with the above property, r must have such atoms in the head.

Suppose the violated rule r is of the form (a). Then $K \setminus H$ contains an atom $A(t)$, for some unary predicate symbol A . It follows that $H \not\models P(U)^K$. This holds since $P(U)^K$ contains $A(t) \leftarrow$ by the definition of local programs.

Consequently, r is of type (b), (c), (d), or (e). Due to $K = I \cap \mathcal{HB}_t$ and the definition of $P(U)$, it follows that $r \in P(U)^K$. Due to $\text{body}(r) \subseteq M$, $M = H \cup (I \setminus K)$, and the atoms that may occur in $\text{body}(r)$, we have $\text{body}(r) \subseteq H$. Furthermore, due to $\text{head}(r) \cap M = \emptyset$, we have $\text{head}(r) \cap H = \emptyset$. This contradicts the assumption that $H \models P(U)^K$.

(B) \Rightarrow (A). Suppose (B) holds, but $I \notin \text{SM}(P)$. Then, $I \notin \text{MM}(P^I)$ and again, there are two possibilities:

- $I \not\models P^I$. Then a rule $r \in P^I$ exists such that $\text{body}(r) \subseteq I$ and $\text{head}(r) \cap I = \emptyset$. Since $\text{ffa}(I) \in \text{SM}(\text{gp}(P))$ and r belongs to the reduct of $\text{gp}(P)$ w.r.t. $\text{ffa}(I)$, r cannot be function-free. Satisfaction of the other rules follows directly from the fact that, for each term $t \hat{\in} I$, $K = I \cap \mathcal{HB}_t$ is a knot that is stable w.r.t. P .
- $I \models P^I$, but is not minimal. Then, some $H \subset I$ exists such that $H \in \text{MM}(P^I)$. Due to forest-shaped model property, H is forest-shaped. If $\text{ffa}(H) \subset \text{ffa}(I)$ would hold, then $\text{ffa}(I) \notin \text{SM}(\text{gp}(P))$ would hold. Therefore, $\text{ffa}(H) = \text{ffa}(I)$ must hold and some term t must exist satisfying the following two conditions.
 - (a) It holds that:

- (I) $A(t) \in I \setminus H$, for some unary predicate symbol A , and t is not a constant, or
 (II) $R(t, s) \in I \setminus H$, for some binary predicate symbol R and a term s ,
 (b) Each subterm v of t violates (a).

Intuitively, t is some smallest term (w.r.t. depth) where I and H disagree on the interpretation of atoms. Suppose t satisfies (I) (and possibly (II)), and is of the form $f(s)$. By assumption, $K = I \cap \mathcal{HB}_s$ is stable w.r.t. P . By choice of t , $K' = H \cap \mathcal{HB}_s$ is a knot such that $K' \subset K$ and $\text{st}(K', s) = \text{st}(K, s)$. As $H \models P^I$, it is easily verified that $K' \models P(\text{st}(K, s))^K$; thus, K is not stable w.r.t. P , a contradiction. Suppose t does not satisfy (I) but satisfies (II). Again, by assumption, $K = I \cap \mathcal{HB}_t$ is stable w.r.t. P . By choice of t and failure of (II), $K' = H \cap \mathcal{HB}_t$ is a knot such that $K' \subset K$ and $\text{st}(K', t) = \text{st}(K, t)$. Again, if $H \models P^I$, then $K' \models P(\text{st}(K, t))^K$; hence K is not stable w.r.t. P , a contradiction.

In both cases we arrive at a contradiction to the assumption that $I \notin SM(P)$. \square

3.2 Finite Representation of Stable Models

By the semantic characterization of the stable models of an FDNC program from above, we may view them as being composed of stable knots. More precisely, we show that Theorem 3.13 allows us to obtain a finite representation of the stable models, which 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 3.14 (KNOT INSTANCE $K_{\downarrow u}$). *Given a term u and a knot K with root term t , the knot $K_{\downarrow u}$ results from K by replacing all occurrences of t in K with u .*

Indeed, if the program P has an infinite stable model I , then the set of knots $L = \{(I \cap \mathcal{HB}_t) \mid t \in I\}$ is infinite. However, for a fixed term t , the set $L' = \{K_{\downarrow t} \mid K \in L\}$ is finite as 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 a common root term, we assume a special constant \mathbf{x} not occurring in any FDNC program. We call a set L of knots \mathbf{x} -grounded, if all its knots have the root term \mathbf{x} . The following notion collects the knots occurring in a stable model and abstracts them using \mathbf{x} .

DEFINITION 3.15 (SCAN $\mathbb{K}(I)$). *Let I be a forest-shaped interpretation for P . We define the set $\mathbb{K}(I)$ of \mathbf{x} -grounded knots as $\mathbb{K}(I) = \{(I \cap \mathcal{HB}_t)_{\downarrow \mathbf{x}} \mid t \in I\}$.*

EXAMPLE 3.16 (CONT'D). *In our bacteria example, for the stable model I (cf. Example 3.2) we have $\mathbb{K}(I) = \{K_7, K_{12}, K_{28}\}$, where each knot is from Figure 3. Note that the maximum term depth in I is 2, and that K_{28} has no child nodes.*

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

PROPOSITION 3.17. *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 .*

$$\begin{aligned}
K_1 &= \emptyset, K_2 = \{W(\mathbf{x})\}, K_3 = \{M(\mathbf{x})\}, K_4 = \{Y(\mathbf{x})\}, K_5 = \{M(\mathbf{x}), Y(\mathbf{x})\}, \\
K_6 &= \{W(\mathbf{x}), Y(\mathbf{x}), Ch(\mathbf{x}, g(\mathbf{x})), M(g(\mathbf{x})), W(g(\mathbf{x}))\} \\
K_7 &= \{W(\mathbf{x}), Y(\mathbf{x}), Ch(\mathbf{x}, g(\mathbf{x})), M(g(\mathbf{x})), C(g(\mathbf{x}))\} \\
K_8 &= \{M(\mathbf{x}), W(\mathbf{x}), Ch(\mathbf{x}, c_1(\mathbf{x})), Y(c_1(\mathbf{x})), W(c_1(\mathbf{x})), Ch(\mathbf{x}, c_2(\mathbf{x})), Y(c_2(\mathbf{x})), W(c_2(\mathbf{x}))\} \\
K_9 &= \{M(\mathbf{x}), Y(\mathbf{x}), W(\mathbf{x}), Ch(\mathbf{x}, c_1(\mathbf{x})), Ch(\mathbf{x}, c_2(\mathbf{x})), \\
&\quad Ch(\mathbf{x}, g(\mathbf{x})), Y(c_1(\mathbf{x})), W(c_1(\mathbf{x})), Y(c_2(\mathbf{x})), W(c_2(\mathbf{x})), M(g(\mathbf{x})), C(g(\mathbf{x}))\}, \\
K_{10} &= \{M(\mathbf{x}), W(\mathbf{x}), Y(\mathbf{x}), Ch(\mathbf{x}, g(\mathbf{x})), Ch(\mathbf{x}, c_1(\mathbf{x})), \\
&\quad Ch(\mathbf{x}, c_2(\mathbf{x})), Y(c_1(\mathbf{x})), Y(c_2(\mathbf{x})), M(g(\mathbf{x})), W(g(\mathbf{x})), W(c_1(\mathbf{x})), W(c_2(\mathbf{x}))\} \\
K_{11} &= \{C(\mathbf{x}), Y(\mathbf{x}), Ch(\mathbf{x}, d(\mathbf{x})), C(d(\mathbf{x})), D(d(\mathbf{x}))\} \\
K_{12} &= \{M(\mathbf{x}), C(\mathbf{x}), Ch(\mathbf{x}, d(\mathbf{x})), C(d(\mathbf{x})), D(d(\mathbf{x}))\} \\
K_{13} &= \{M(\mathbf{x}), C(\mathbf{x}), Y(\mathbf{x}), Ch(\mathbf{x}, d(\mathbf{x})), C(d(\mathbf{x})), D(d(\mathbf{x}))\} \\
K_{14} &= \{C(\mathbf{x}), Ch(\mathbf{x}, d(\mathbf{x})), C(d(\mathbf{x})), D(d(\mathbf{x}))\} \\
K_i &= K_{i-14} \cup \{D(\mathbf{x})\}, i = 15, \dots, 24 \\
K_j &= K_{j-14} \cup \{D(\mathbf{x})\} \setminus \{Ch(\mathbf{x}, d(\mathbf{x})), C(d(\mathbf{x})), D(d(\mathbf{x}))\}, j = 25, \dots, 27 \\
K_{28} &= \{C(\mathbf{x}), D(\mathbf{x})\}
\end{aligned}$$

Fig. 3. All stable \mathbf{x} -grounded knots of the bacteria program

EXAMPLE 3.18 (CONT'D). Recall that the knots K_1 and K_2 in Figure 2 are stable w.r.t. P , and so are their \mathbf{x} -grounded versions. In total, there exist 28 \mathbf{x} -grounded knots that are stable w.r.t. P , which are shown in Figure 3.

We introduce a 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., the terms t and s satisfy the same unary predicates.

DEFINITION 3.19 (FOUNDED KNOT SET). Let L be a set of \mathbf{x} -grounded knots. Then L is *founded* w.r.t. a program P and a set of states S , if the following hold:

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

EXAMPLE 3.20 (CONT'D). In our example, the set of all \mathbf{x} -grounded stable knots (see Figure 3) is founded w.r.t. P and $S = \{\text{st}(I) \mid I \text{ is an interpretation of } P\}$. Indeed, for any interpretation I and $c \hat{\in} I$, some knot K_i exists such that $\text{st}(I, c) \approx \text{st}(K, \mathbf{x})$; hence, Condition 1) is satisfied. As easily seen, Condition 2) also holds.

The following is easy to verify (recall $\text{st}(I)$ from Definition 3.5).

PROPOSITION 3.21. Let $I \in SM(P)$. Then $\mathbb{K}(I)$ is a set of knots that is founded w.r.t. P and $\text{st}(I)$.

EXAMPLE 3.22 (CONT'D). Recall that for the stable model I (cf. Example 3.2) we have $\mathbb{K}(I) = \{K_7, K_{12}, K_{28}\}$. It is easily checked that $\mathbb{K}(I)$ is founded w.r.t. P and $\text{st}(I)$, which contains the single state $\{\text{Young}(b), \text{Warm}(b)\}$: the knot K_7 satisfies condition 1), and considering K_7, K_{12} , and K_{28} in this order we can verify condition 2) (note that $\text{succ}(K_{28}) = \emptyset$).

In what follows, we give a construction of stable models out of knots in a founded set. Moreover, we characterize the set of stable models via founded knot sets.

Generating Stable Models using Knots. To construct stable models as forest-shaped interpretations from knots in a founded knot set, we start with constructing respective trees, which are represented as usual by prefix-closed sets of words. For a sequence of elements $p = [e_1, \dots, e_n]$, let $\tau(p)$ denote the last element e_n , and $[p|e_{n+1}]$ denote the sequence $[e_1, \dots, e_n, e_{n+1}]$.

DEFINITION 3.23 (TREE CONSTRUCTION). Given a set L of \mathbf{x} -grounded knots and a state U^t , a set T of sequences $[e_1, \dots, e_n]$, whose elements $e_i = \langle K_i, t_i \rangle$ are pairs of knots K_i and terms t_i , is a tree induced by L with root state U^t , if:

- (a) T contains some $[\langle K, t \rangle]$ s.t. $K \in L$ and $\text{st}(K, \mathbf{x}) \approx U^t$.
- (b) For every $p \in T$ with $\tau(p) = \langle K, t \rangle$ and $f(\mathbf{x}) \in \text{succ}(K)$, T contains some $[p|\langle K', f(t) \rangle]$ s.t. $K' \in L$ and $\text{st}(K, f(\mathbf{x})) \approx \text{st}(K', \mathbf{x})$.
- (c) T is minimal, i.e., each $T' \subset T$ violates (a) or (b).

Intuitively, each path $p \in T$ is a node in the tree. If $\tau(p) = \langle K, t \rangle$, then p represents the term t and the \mathbf{x} -grounded knot K ; to obtain an interpretation, K will be instantiated with t . To obtain stable models, we require for closure under successor knots (see 3.a in Definition 3.19) which is achieved via (b) above.

EXAMPLE 3.24 (CONT'D). Let $L = \mathbb{K}(I)$ for the stable model I of P in Example 3.2. Then the tree $T = \{[\langle K_7, b \rangle], [\langle K_7, b \rangle, \langle K_{12}, \text{grow}(b) \rangle], [\langle K_7, b \rangle, \langle K_{12}, \text{grow}(b) \rangle, \langle K_{28}, \text{die}(\text{grow}(b)) \rangle]\}$ is induced by L with root state $\{\text{Young}(b), \text{Warm}(b)\} \approx \text{st}(K_7, \mathbf{x})$.

A tree T induced by some \mathbf{x} -grounded knot set L with root state U^t is transformed into a set of ground atoms defined by $T_\downarrow = \bigcup \{K_{\downarrow t} \mid p \in T \text{ with } \tau(p) = \langle K, t \rangle\}$. This is generalized to collections of trees whose roots are connected as follows.

DEFINITION 3.25 (FOREST MODEL CONSTRUCTION). Let G be a set of function-free ground atoms and let L be a set of knots founded w.r.t. P and a set of states $S \supseteq \text{st}(G)$. Then $\mathcal{F}(G, L)$ is the largest set of forest-shaped interpretations

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

where $\{c_1, \dots, c_n\}$ is the set of all constants occurring in G and each T^{c_i} is a tree induced by L with root state $\text{st}(G, c_i)$.

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

THEOREM 3.26. If $G \in \text{SM}(\text{gp}(P))$, and L is a set of knots that is founded w.r.t. P and some $S \supseteq \text{st}(G)$, 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 3.17, for each term $t \in I$, $I \cap \mathcal{HB}_t$ is a knot that is stable w.r.t. P . Keeping in mind that $G \in SM(\mathbf{gp}(P))$, Theorem 3.13 implies that I is a stable model of P . \square

EXAMPLE 3.27 (CONT'D). *The set $G = \{\text{Young}(b), \text{Warm}(b)\}$ is the single stable model of $\mathbf{gp}(P)$, and $\mathbb{K}(I) = \{K_7, K_{12}, K_{28}\}$ is founded w.r.t. P and $\text{st}(I) = \text{st}(G)$ ($= \{G\}$) for the stable model I of P . The tree T in Example 3.24 is induced by $\mathbb{K}(I)$, and in fact it is the only tree induced by $\mathbb{K}(I)$ with root state G . Hence, $\mathcal{F}(G, \mathbb{K}(I))$ contains the single interpretation $G \cup (T^b)_\downarrow$, which coincides with I .*

We showed that stable model existence can be proved by checking that some suitable founded knot set 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. The following property of founded knot sets is obvious.

PROPOSITION 3.28. *Let L_1 and L_2 be sets of knots founded w.r.t. P and sets of states S_1 and S_2 , respectively. Then $L_1 \cup L_2$ is founded w.r.t. P and $S_1 \cup S_2$.*

At this point, we introduce a founded set of knots, which will capture all the stable models. Recall $\text{st}(P)$ from Definition 3.5.

DEFINITION 3.29 (\mathbb{K}_P). *We denote by \mathbb{K}_P the smallest set of knots which contains every set of knots L that is founded w.r.t. P and some $S \subseteq \text{st}(\mathbf{gp}(P))$.*

Due to Proposition 3.28 and Definition 3.29, the following is immediate.

PROPOSITION 3.30. *For the program P , the following hold:*

- (a) \mathbb{K}_P is founded w.r.t. P and some $S \subseteq \text{st}(\mathbf{gp}(P))$.
- (b) If L is a set of knots that is founded w.r.t. P and some $S \subseteq \text{st}(\mathbf{gp}(P))$, 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 , for every $S \subseteq \text{st}(\mathbf{gp}(P))$.

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 3.19.

EXAMPLE 3.31 (CONT'D). *In our bacteria example, $\mathbb{K}_P = \{K_6, K_7, K_8, K_{12}, K_{28}\}$. Note that \mathbb{K}_P contains besides the knots K_7, K_{12}, K_{28} in $\mathbb{K}(I)$ for the stable model I in Example 3.2 also the knots K_6 and K_8 ; the initial part of the stable model shown in Figure 1 is built using instances of K_6 and K_8 .*

PROPOSITION 3.32. *If $I \in SM(P)$, then $I \in \mathcal{F}(\text{ffa}(I), L)$ for every set of knots $L \supseteq \mathbb{K}(I)$ that is founded w.r.t. P and some state set $S \supseteq \text{st}(I)$.*

The following will be helpful.

DEFINITION 3.33 (COMPATIBLE \mathbb{K}_P). *We call \mathbb{K}_P compatible with a set of states S , if for every state $U \in S$ some $K \in \mathbb{K}_P$ exists s.t. $U \approx \text{st}(K, \mathbf{x})$.*

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 $\mathbf{gp}(P)$, it represents the latter.

| Problem | F | FD | FC | FDC, FN, FNC, FDNC |
|---|--------------|--------------------|--------------|--------------------|
| Consistency | Trivial | Trivial | PSPACE (6.2) | EXPTIME (5.2, 6.1) |
| $P \models_b A(\vec{t})$ | P (6.3) | Σ_2^P (6.3) | PSPACE (6.2) | EXPTIME (5.3) |
| $P \models_b \exists \vec{x}. A(\vec{x})$ | PSPACE (6.3) | PSPACE (6.3) | PSPACE (6.2) | EXPTIME (5.3) |
| $P \models_c A(\vec{t})$ | P (6.3) | co-NP (6.3) | PSPACE | EXPTIME |
| $P \models_c \exists \vec{x}. A(\vec{x})$ | PSPACE | EXPTIME | PSPACE | EXPTIME |
| $P \models_c \lambda \vec{x}. A(\vec{x})$ | PSPACE | EXPSpace (5.4) | PSPACE | EXPSpace (5.4) |

Table I. Complexity of FDNC and Fragments (Completeness Results)

THEOREM 3.34. *Let I be an interpretation for P . Then, $I \in SM(P)$ iff $I \in \mathcal{F}(G, \mathbb{K}_P)$, for some $G \in SM(\mathbf{gp}(P))$ such that \mathbb{K}_P is compatible with $\mathbf{st}(G)$.*

PROOF. If $I \in SM(P)$, then, by Proposition 3.21, $\mathbb{K}(I)$ is founded w.r.t. P and $\mathbf{st}(I)$. By definition, $\mathbb{K}(I) \subseteq \mathbb{K}_P$. By Proposition 3.30, \mathbb{K}_P is founded w.r.t. P and some $S \supseteq \mathbf{st}(I)$. By Proposition 3.32, $I \in \mathcal{F}(\mathbf{ffa}(I), \mathbb{K}_P)$. Note that $\mathbf{ffa}(I) \in SM(\mathbf{gp}(P))$. The other direction is proved by Theorem 3.26. \square

We have obtained a finite representation of the stable models of an FDNC program P . Indeed, each of its stable models can be generated out of some stable model of $\mathbf{gp}(P)$ and the knot set \mathbb{K}_P .

EXAMPLE 3.35 (CONT'D). *From $\mathbb{K}_P = \{K_6, K_7, K_8, K_{12}, K_{28}\}$ and the only stable model $G = \{\mathit{Young}(b), \mathit{Warm}(b)\}$ of $\mathbf{gp}(P)$, we can construct the stable model I from Example 3.2, as well as any other stable model of P .*

We can view $\mathbf{gp}(P)$ together with \mathbb{K}_P as a compilation of the program P that can be exploited for reasoning and stable model building (see Sections 5 and 6).

4. COMPLEXITY RESULTS

This section gives a brief overview of our results on the complexity of the main reasoning tasks in FDNC and its fragments, which are compactly summarized in Table I. An in-depth analysis and the reasoning techniques for the derivation are given in the following two sections. Here, we give some intuition behind the results and discuss how some of them can be derived from a core of results.

As shown in the previous section, FDNC programs have forest-shaped stable models. Naturally, reasoning in FDNC involves construction of forest-shaped interpretations (in the following, *forests*). Consistency testing involves building a forest-shaped stable model, while brave/cautious reasoning requires checking whether some property holds in some/all stable models that can be built. However, an FDNC program may have infinite stable models, and therefore the construction has to employ some direct or indirect blocking technique to stop the construction after sufficient information is acquired.

The forest-shaped model property implies that blocking of the model construction is feasible and, hence, the decidability of FDNC for major reasoning tasks can be established. Indeed, a continuous construction of a forest will lead to reoccurrences of patterns, e.g., states of terms, non-isomorphic labeled arcs, trees of depth 1, etc. To find algorithms for FDNC, we could resort to methods of Description Logics (DLs), which often have the forest-shaped model property and are decided

by tableau methods with blocking. Unfortunately, such methods are not well-suited in our case. First, they cannot easily handle minimality testing and are generally not worst-case optimal. Second, tableau methods are designed for consistency testing, while some important tasks from non-monotonic reasoning (e.g. brave reasoning) cannot always be polynomially reduced to consistency testing (see Table I).

Therefore, our algorithms for \mathbb{FDNC} rely on the finite representation of stable models in terms of maximal founded sets of knots. In Section 5.1, we show how to derive the set \mathbb{K}_P of knots for a given \mathbb{FDNC} program P in single exponential time in the size of P . This is possible as the number of distinct \mathbf{x} -grounded knots is bounded by a single exponential. Given \mathbb{K}_P , several standard reasoning tasks can be solved in time polynomial in the size of \mathbb{K}_P ; hence, overall they are in EXPTIME . This includes consistency testing (Section 5.2), brave entailment of ground and existential queries (Section 5.3), as well as cautious entailment of ground and existential queries (which is easily reduced to consistency testing). These upper bounds are tight for \mathbb{FDNC} . It is easy to see that a decision procedure needs to explore forests whose depths are bounded by a single exponential in the size of the input program. However, due to the disjunction or non-monotonic negation in an \mathbb{FDNC} program, the number of such candidate forests may be too high for a procedure to traverse them in polynomial space. The EXPTIME -hardness of consistency testing already in \mathbb{FDC} is proved in Section 5.2 by an encoding of an EXPTIME -hard Description Logic \mathcal{ALC} , which is extended to \mathbb{FN} in Section 6.1. The hardness of consistency testing directly provides lower bounds for brave and cautious entailment of ground and existential queries. We note also that the EXPTIME -completeness results for \mathbb{FDC} and \mathbb{FN} show that these fragments are equal in terms of problem solving capacity: unlike in the propositional setting (cf. [Dantsin et al. 2001]), negation alone can polynomially compensate disjunction and constraints, and vice versa.

For the fragment \mathbb{FC} of \mathbb{FDNC} , the picture is different. Its programs have the unique model property, i.e., if a stable model exists, it is unique. For the standard reasoning tasks, a procedure thus needs to navigate a unique forest searching for a node with a certain property, e.g., the one that causes an inconsistency, or satisfies a query. Furthermore, the procedure needs to navigate only the depths bounded by a single exponential. Our algorithms navigate the forest by non-deterministically guessing the paths through function symbols and building necessary parts of a stable model. They run in polynomial space and can, by Savitch's Theorem [1970], turned into deterministic polynomial space algorithms. The PSPACE -hardness of consistency testing is shown by a Turing machine encoding, which is extended to other standard reasoning tasks (see Section 6.2).

If we disallow non-monotonic negation and constraints, the complexity drops even more. Consistency testing in both \mathbb{F} and \mathbb{FD} is trivial, while the complexity of ground entailment drops to lower levels of the polynomial hierarchy, and corresponds to the complexity of propositional logic programming. This is because consistency needs not be ensured, and the necessary conditions can be verified locally within polynomial distance from the graph part of the input program. Section 6.3 discusses the results for \mathbb{F} and \mathbb{FD} .

The last row in Table I lists the complexity of open queries. Deciding cautious entailment of open queries in \mathbb{FDNC} is EXPSpace -complete and thus harder than

cautious entailment of existential queries. Intuitively, this is because to search for a term that satisfies a property in each stable model of a program, we must look at branches beyond single exponential length. However, the length can be bounded by a double exponential, and we can thus manage to answer the query in single exponential space; Section 5.4 provides the details. As noted in the preliminaries, in case of brave entailment, the semantics of open and existential queries coincide, and hence the complexity results on existential queries carry over to open queries.

The main entries in Table I are presented with references to the sections that discuss the respective problems in detail. The other entries are justified as follows:

- (1) Programs in \mathbb{F} and \mathbb{FD} are positive and without constraints, hence consistent.
- (2) PSPACE-hardness (resp. EXPTIME-hardness) of $P \models_c \exists \vec{x}. A(\vec{x})$ in \mathbb{F} and \mathbb{FC} (resp. in \mathbb{FD} and \mathbb{FDC}) holds as consistency checking with constraints in \mathbb{FC} (resp. \mathbb{FDC}) is reducible to cautious inference. On the other hand, completeness also holds as cautious inference is reducible to inconsistency testing in the standard way.
- (3) Similarly, PSPACE (resp. EXPTIME) membership of $P \models_c A(\vec{t})$, where P is a program in \mathbb{FC} (resp. \mathbb{FDC} , \mathbb{FN} , \mathbb{FNC} or \mathbb{FDNC}), holds as the problem amounts to checking consistency of $P \cup \{\leftarrow A(\vec{t})\}$. On the other hand, hardness holds as P is inconsistent iff $P \models_c A'(t)$, where A' is a fresh symbol and t is arbitrary.
- (4) PSPACE-completeness of $P \models_c \lambda \vec{x}. A(\vec{x})$ in \mathbb{F} and \mathbb{FC} holds because these fragments have the unique stable model property, and hence cautious entailment of open and existential queries coincide; the latter is PSPACE-complete.

To ease presentation, we use a lemma that allows us to focus on unary queries.

LEMMA 4.1. *Let C be a complexity class in Table I, and let \mathcal{L} be from the \mathbb{F} family. Then:*

- (i) *If deciding program consistency for \mathcal{L} is C -hard, then deciding brave entailment of queries (ground or existential, unary or binary) is also C -hard for \mathcal{L} .*
- (ii) *Brave entailment of unary existential (resp., ground) queries is C -complete for \mathcal{L} iff brave entailment of binary existential (resp., ground) queries is C -complete for \mathcal{L} .*
- (iii) *Cautious entailment of unary open queries is C -complete for \mathcal{L} iff cautious entailment of binary open queries is C -complete for \mathcal{L} .*

For a proof of Lemma 4.1, we refer to the appendix. Intuitively, the first statement holds as brave reasoning involves a construction of a stable model containing a certain atom, which cannot be easier than constructing (or checking the existence of) an arbitrary stable model. The second statement hinges on the fact that the class \mathbb{F} allows rules $A(y) \leftarrow R(x, y)$ and $R(x, f(x)) \leftarrow A(x)$, by which brave entailment of binary queries can be reformulated in terms of unary queries, and vice versa. Similarly, with rules in the syntax of \mathbb{F} , one constructs reductions proving (iii).

5. COMPLEXITY OF \mathbb{FDNC}

This section discusses the complexity of reasoning in \mathbb{FDNC} and provides worst-case optimal algorithms together with the matching hardness results. The methods for consistency testing, deciding brave entailment of ground and existential queries

and cautious entailment of open queries rely on the finite representation of stable models in terms of the set \mathbb{K}_P of knots which, together with the set $SM(\text{gp}(P))$, captures all the stable models of an FDNC program P (see Theorem 3.34).

5.1 Deriving Maximal Founded Set of Knots

To derive \mathbb{K}_P , we proceed in two phases. In the first phase, we generate the set of knots that surely contains \mathbb{K}_P . In the second phase, we remove some knots from it to ensure that it satisfies Definition 3.29.

To ease the presentation, for any knot set L , let $\text{st}^{+1}(L) = \{\text{st}(K, s) \mid K \in L, s \in \text{succ}(K)\}$, i.e., $\text{st}^{+1}(L)$ is the set of all states of the successor terms of knots in L .

DEFINITION 5.1 ($\text{All}(P)$). *For an FDNC program P , let $\text{All}(P)$ be the smallest set of \mathbf{x} -grounded knots satisfying the following conditions:*

- a) *If $U \in \text{st}(\text{gp}(P))$ and $K \in SM(P(U))$, then $K \downarrow_{\mathbf{x}} \in \text{All}(P)$.*
- b) *If $U \in \text{st}^{+1}(\text{All}(P))$ and $K \in SM(P(U))$, then $K \downarrow_{\mathbf{x}} \in \text{All}(P)$.*

Intuitively, $\text{All}(P)$ contains by construction each set of knots that is founded w.r.t. P and some set of states $S \subseteq \text{st}(P)$. The first problem is that $\text{All}(P)$ might contain knots K that lack some successor knots, i.e., for some $s \in \text{succ}(K)$ no K' is in $\text{All}(P)$ s.t. $\text{st}(K, s) \approx \text{st}(K', \mathbf{x})$ (condition (3.a) in Definition 3.19). On the other hand, each knot in \mathbb{K}_P must be reachable from a state in $\text{st}(P)$ (condition (3.b)). These requirements are ensured by removing some knots from $\text{All}(P)$.

DEFINITION 5.2 ($\text{bad}(L)$). *For any set of \mathbf{x} -grounded knots L , $\text{bad}(L)$ is the smallest subset of L such that $K \in \text{bad}(L)$, if for some $s \in \text{succ}(K)$, either*

- a) *no $K' \in L$ fulfills $\text{st}(K, s) \approx \text{st}(K', \mathbf{x})$, or*
- b) *for all $K' \in L$, $\text{st}(K, s) \approx \text{st}(K', \mathbf{x})$ implies $K' \in \text{bad}(L)$.*

Intuitively, obtaining the set $\text{All}(P) \setminus \text{bad}(\text{All}(P))$ corresponds to iteratively removing from $\text{All}(P)$ the knots that have no successors (note that removing a knot from a set L might leave some other knots in L without a successor).

The following notion will help to ensure satisfaction of (3.b) of Definition 3.19.

DEFINITION 5.3 ($\text{reach}_S(L)$). *For any set of \mathbf{x} -grounded knots L and set of states S , $\text{reach}_S(L)$ is the smallest set of knots such that:*

- a) *if $U \in S$, $K \in L$ and $U \approx \text{st}(K, \mathbf{x})$, then $K \in \text{reach}_S(L)$, and*
- b) *if $U \in \text{st}^{+1}(\text{reach}_S(L))$, $K \in L$ and $U \approx \text{st}(K, \mathbf{x})$, then $K \in \text{reach}_S(L)$.*

Intuitively, $\text{reach}_S(L)$ are the knots in L reachable from the states in S . Indeed, if $\text{reach}_S(L) = L$, then L fulfills condition (3.b) of Definition 3.19 w.r.t. S .

THEOREM 5.4. *If P is an FDNC program, then $\mathbb{K}_P = \text{reach}_{\text{st}(\text{gp}(P))}(L_P)$, where $L_P = \text{All}(P) \setminus \text{bad}(\text{All}(P))$.*

PROOF. Let $L = \text{reach}_{\text{st}(\text{gp}(P))}(\text{All}(P) \setminus \text{bad}(\text{All}(P)))$. We verify that L satisfies the conditions in Definition 3.29, i.e., L is the single \subseteq -minimal set which contains each knot set L' that is founded w.r.t. P and some $S \subseteq \text{st}(\text{gp}(P))$.

Indeed, every such L' fulfills $L' \subseteq \text{All}(P)$; by construction of L , no $K \in L'$ is removed, thus $L' \subseteq L$. To prove the result, it is thus sufficient to show that L itself

is founded w.r.t. P and some $S \subseteq \text{st}(\text{gp}(P))$ (cf. Definition 3.19). The definition of $\text{All}(P)$ ensures that every $K \in L$ is stable w.r.t. P . Furthermore, the removal of knots in $\text{bad}(\text{All}(P))$ and restriction to reachable knots ensures that every $K \in L$ has proper successors as in (3.a) of Definition 3.19, and has a proper predecessor sequence as in (3.b) that reaches a state in $\text{st}(\text{gp}(P))$ (we can set S suitably). \square

It is easy to see that we can compute \mathbb{K}_P in time at most single exponential in the size of an FDNC program P . This is immediate from the following observations:

- The number of \mathbf{x} -grounded knots over P is bounded by a single exponential in the size of P . More precisely, the number is bounded by $b(P) = 2^{n+k \cdot (n+m)}$, when P has k function, n unary, and m binary predicate symbols.
- Computing $\text{All}(P)$ requires adding at most $b(P)$ \mathbf{x} -grounded knots. Each such knot has polynomial size and its stability is verifiable using a $\Sigma_2^P = \text{NP}^{\text{NP}}$ oracle. Thus, $\text{All}(P)$ is computable in time single exponential in the size of P .
- Computing $\text{bad}(L)$ is polynomial in the size of L . Thus, $\text{All}(P) \setminus \text{bad}(\text{All}(P))$ is computable in time single exponential in the size of P .
- The size of $\text{st}(\text{gp}(P))$ is bounded by a single exponential in the size of P .
- Computing $\text{reach}_S(L)$ is polynomial in the combined size of L and S .

5.2 Deciding Consistency

Once the set \mathbb{K}_P for an FDNC program P is derived, it can be readily used for consistency testing. We will see that the resulting algorithm is worst-case optimal.

THEOREM 5.5. *For every FDNC program P , the following are equivalent:*

- (i) P is consistent.
- (ii) For some $G \in \text{SM}(\text{gp}(P))$, the set \mathbb{K}_P is compatible with $\text{st}(G)$.

PROOF. If $I \in \text{SM}(P)$, then by Theorem 3.34 some $G \in \text{SM}(\text{gp}(P))$ exists such that \mathbb{K}_P is compatible with $\text{st}(G)$. The converse is proved by Theorem 3.26. \square

By this theorem, to decide consistency of P we can search for a stable model G of the program $\text{gp}(P)$ such that for each constant of P , \mathbb{K}_P can start the tree construction (i.e., \mathbb{K}_P is compatible with $\text{st}(G)$). We obtain the following result.

THEOREM 5.6. *Deciding whether an FDNC program is consistent is in EXPTIME.*

PROOF. Deciding whether \mathbb{K}_P is compatible with $\text{st}(G)$, for some $G \in \text{SM}(\text{gp}(P))$, is feasible in time polynomial in $n + m$, where m is the size of \mathbb{K}_P and n is the size of $\text{SM}(\text{gp}(P))$. Overall, this can be done in time single exponential in the size of P , since both m and n are single exponential in the size of P . Since $\text{SM}(\text{gp}(P))$ is computable in single exponential time, the result follows from Theorem 5.5. \square

As we have pointed earlier already, we can see \mathbb{K}_P together with $\text{gp}(P)$ as a compilation of the FDNC program P . Out of this compilation, we can gradually build a stable model of P by continuing the tree construction for some stable model G of $\text{gp}(P)$ using knots from \mathbb{K}_P (and every stable model of P results by proper choices). Here the hard part is computing such a G , which depending on the complexity of function-free programs is Σ_2^P -hard already for FD , NP -hard already for FN , and polynomial for F and FC . Checking the compatibility of \mathbb{K}_P with

| | |
|--|--|
| Mapping knowledge base \mathcal{K} | |
| $\Theta(\mathcal{K}) = \bigcup_{\alpha \in \mathcal{K}} \{\Pi(\alpha)\}$ | |
| Mapping axioms and assertions | |
| $\Pi(C(a)) = p_C(a)$ | $\Pi(R(a, b)) = p_R(a, b)$ |
| $\Pi(C \sqsubseteq D) = (\forall x)(\pi(C, x) \rightarrow \pi(D, x))$ | |
| Mapping concepts | |
| $\pi(\top, X) = \top$ | $\pi(C \sqcup D, X) = \pi(C, X) \vee \pi(D, X)$ |
| $\pi(A, X) = p_A(X)$ | $\pi(C \sqcap D, X) = \pi(C, X) \wedge \pi(D, X)$ |
| $\pi(\perp, X) = \perp$ | $\pi(\forall R.C, X) = (\forall y)(p_R(X, y) \rightarrow \pi(C, y))$ |
| $\pi(\neg C, X) = \neg\pi(C, X)$ | $\pi(\exists R.C, X) = (\exists y)(p_R(X, y) \wedge \pi(C, y))$ |

Note: X is a meta-variable that is replaced by an actual variable.

Fig. 4. Semantics of the DL \mathcal{ALC} by mapping to first-order logic

$\text{st}(G)$ is polynomial, and each tree expansion step using a knot from \mathbb{K}_P is feasible with low (clearly polynomial) cost. Note that this model-building technique is complementary to computing a stable model of an ordinary (function-free) logic program, and may be realized on top of stable model engines like DLV or Smodels.

In the following, we show that the algorithm emerging from Theorem 5.5 is worst-case optimal. The proof is by a polynomial-time translation of consistency testing in the Description Logic \mathcal{ALC} , which is EXPTIME-hard, to consistency testing in \mathbb{FDC} . The translation is interesting in its own right, as it provides a translation of the core of expressive Description Logics into logic programming.

DEFINITION 5.7 (\mathcal{ALC} SYNTAX). Let $\mathbf{C} \supseteq \{\top, \perp\}$, \mathbf{R} and \mathbf{I} denote the sets of concept, role, and individual names, respectively. Inductively, each $C \in \mathbf{C}$ is a concept, and if C, D are concepts and R is a role, then $C \sqcap D$, $C \sqcup D$, $\neg C$, $\forall R.C$, and $\exists R.C$ are concepts. If $C \in \mathbf{C}$, then C and $\neg C$ are literal concepts.

A general concept inclusion axiom (GCI) is of form $C \sqsubseteq D$ where C, D are concepts. An assertion is of form $C(a)$ or $R(a, b)$, where $a, b \in \mathbf{I}$, $R \in \mathbf{R}$, and $C \in \mathbf{C}$. An \mathcal{ALC} knowledge base is a finite set of GCIs and assertions.

Each \mathcal{ALC} knowledge base \mathcal{K} has a model-theoretic semantics, which is given via a mapping of \mathcal{K} into a set of sentences in first-order logic, shown in Figure 4 (see e.g. [Hustadt et al. 2004] for details). The major reasoning task in \mathcal{ALC} is deciding the *consistency* of a given \mathcal{K} , i.e., that of the first-order theory $\Theta(\mathcal{K})$.

We now provide a polynomial time translation of *normalized* \mathcal{ALC} knowledge bases \mathcal{K} into \mathbb{FDC} programs $P^{\mathcal{K}}$ such that \mathcal{K} is consistent iff $P^{\mathcal{K}}$ is consistent. Normalized knowledge bases obey certain structural constraints which makes presenting the translation easier.

DEFINITION 5.8 (\mathcal{ALC} NORMAL FORM). An \mathcal{ALC} knowledge base \mathcal{K} is in normal form, if its GCI axioms are of one of the following forms:

$$\begin{array}{ll}
(T1) & A_0 \sqcap \dots \sqcap A_n \sqsubseteq B_0 \sqcup \dots \sqcup B_m, & (T4) & A_0 \sqsubseteq \exists R.B_0, \\
(T2) & A_0 \sqcap \dots \sqcap A_n \sqsubseteq \perp, & (T5) & A_0 \sqsubseteq \forall R.B_0, \\
(T3) & \top \sqsubseteq B_0 \sqcup \dots \sqcup B_m, & &
\end{array}$$

| Axioms of \mathcal{K} | Rules of $P^{\mathcal{K}}$ |
|---|--|
| (T1) $A_0 \sqcap \dots \sqcap A_n \sqsubseteq B_0 \sqcup \dots \sqcup B_m$ | $B_0(x) \vee \dots \vee B_m(x) \leftarrow A_0(x), \dots, A_n(x)$ |
| (T2) $A_0 \sqcap \dots \sqcap A_n \sqsubseteq \perp$ | $\leftarrow A_0(x), \dots, A_n(x)$ |
| (T4) $A \sqsubseteq \exists R.C$ | $R'(x, f(x)) \leftarrow A(x)$ $R(x, y) \leftarrow R'(x, y)$ $C(y) \leftarrow R'(x, y)$ |
| (T5) $A \sqsubseteq \forall R.C$ | $C(y) \leftarrow A(x), R(x, y)$ |
| $A(a)$ | $A(a) \leftarrow;$ |
| $R(a, b)$ | $R(a, b) \leftarrow;$ |
| where $n \geq 0$, f is fresh function symbol, R' is a fresh binary predicate symbol. | |

Table II. Translating \mathcal{ALC} into $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$

where $n, m \geq 0$, and each A_i and B_j is from \mathbf{C} , but is neither \top nor \perp .³ Moreover, if \mathcal{K} is in normal form and does not contain axioms of type (T3), then \mathcal{K} is safe.⁴

Importantly, we can normalize any \mathcal{ALC} knowledge base \mathcal{K} efficiently.

PROPOSITION 5.9. *Given any \mathcal{ALC} knowledge base \mathcal{K} , we can obtain in linear time a safe knowledge base \mathcal{K}' in normal form that is consistent iff \mathcal{K} is consistent.*

The proof, which is based on well-known *definitional form transformations*, is given in the appendix. We are now ready to define the translation. For any safe knowledge base \mathcal{K} in normal form, let $P^{\mathcal{K}}$ denote the $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ program that results after applying the translation rules in Table II.

PROPOSITION 5.10. *Let \mathcal{K} be a safe knowledge base in normal form. Then \mathcal{K} is consistent iff $P^{\mathcal{K}}$ is consistent.*

PROOF. It is easy to verify that $P^{\mathcal{K}}$ is a rule-representation of the first-order theory that is obtained from $\Theta(\mathcal{K})$ by applying Skolemization and a satisfiability preserving transformation for the axioms of type (T4). By Herbrand's Theorem [1971], $P^{\mathcal{K}}$ is consistent iff $\Theta(\mathcal{K})$ consistent. \square

Note that $P^{\mathcal{K}}$ is in fact positive and constructible in linear time from \mathcal{K} . Hence, Propositions 5.9–5.10 and the well-known EXPTIME -hardness of \mathcal{ALC} [Schild 1991] imply that deciding consistency of $\mathbb{F}\mathbb{D}\mathbb{C}$ and $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ programs is EXPTIME -hard. Combined with Theorem 5.6, we establish the completeness result.

THEOREM 5.11. *Checking consistency of $\mathbb{F}\mathbb{D}\mathbb{C}$ and $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ programs is EXPTIME -complete.*

5.3 Brave Entailment of Queries

As we did for consistency checking, we exploit the set \mathbb{K}_P for a program P to provide algorithms for brave reasoning. We first discuss entailment of existential unary atomic queries, and then of ground queries. The idea behind the method is to perform some “back-propagation” of unary predicate symbols in the set of knots.

³For a similar normal form for the weaker Description Logic \mathcal{EL}^{++} , see [Baader et al. 2005].

⁴We require safety because applying the transformation Π to an axiom $\top \sqsubseteq B_0 \sqcup \dots \sqcup B_m$ leads to the formula $\forall x.B_0(x) \vee \dots \vee B_m(x)$ which, in turn, cannot be stated as a safe rule in $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ syntax.

DEFINITION 5.12 ($\text{reach}_A(L)$). For every set L of \mathbf{x} -grounded knots and unary predicate symbol A , let $\text{reach}_A(L)$ be the smallest subset of L such that:

- (a) if $K \in L$ and $A(\mathbf{x}) \in K$, then $K \in \text{reach}_A(L)$, and
- (b) if $K' \in \text{reach}_A(L)$ is a possible successor of $K \in L$, i.e., for some $s \in \text{succ}(K)$ we have $\text{st}(K, s) \approx \text{st}(K', \mathbf{x})$, then $K \in \text{reach}_A(L)$.

Intuitively, $K \in \text{reach}_A(L)$ means that, starting from K , a sequence of possible successor knots will eventually reach a knot containing $A(\mathbf{x})$. Since \mathbb{K}_P together with $SM(\text{gp}(P))$ captures the stable models of P , we have the following:

THEOREM 5.13. For every program P , the following statements are equivalent:

- (A) $P \models_b \exists x.A(x)$.
- (B) Some $G \in SM(\text{gp}(P))$ exists such that (i) \mathbb{K}_P is compatible with $\text{st}(G)$, and (ii) for some constant c and $K \in \mathbb{K}_P$, $\text{st}(G, c) \approx \text{st}(K, \mathbf{x})$ and $K \in \text{reach}_A(\mathbb{K}_P)$.

Theorem 5.13 provides us with an algorithm, since brave entailment of existential queries can be decided by verifying the condition (B) of the theorem. As easily seen, the condition is verifiable in time single exponential in the size of the program P . Indeed, computing $\text{reach}_A(\mathbb{K}_P)$ requires time quadratic in the size of \mathbb{K}_P , or single exponential in the size of P . Once \mathbb{K}_P , $\text{reach}_A(\mathbb{K}_P)$, and $SM(\text{gp}(P))$ are computed, the conditions in (B) are verifiable in time polynomial in the combined size of \mathbb{K}_P , $\text{reach}_A(\mathbb{K}_P)$, and $SM(\text{gp}(P))$. Hence, (B) can be verified in time single exponential in the size of P , that is, for a given FDNC program, the problem of deciding whether it bravely entails a unary existential query is in EXPTIME. On the other hand, due to Theorem 5.11 and Lemma 4.1, we know that brave entailment of unary existential queries is EXPTIME-hard already for FDC. Therefore, we conclude the following.

THEOREM 5.14. For FDC and FDNC programs, brave entailment of an existential unary query is EXPTIME-complete. The same holds for binary existential queries (see Lemma 4.1).

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

DEFINITION 5.15 ($\text{goal}_q(L)$). Let $q = A(t)$ be a ground atom and L be a set of \mathbf{x} -grounded knots. Let T be the set of subterms of the term t . Then $\text{goal}_q(L)$ is the smallest relation over $L \times T$ such that:

- (a) if $K \in L$ and $A(\mathbf{x}) \in K$, then $\langle K, t \rangle \in \text{goal}_q(L)$, and
- (b) if there exist (i) $K \in L$ with $f(\mathbf{x}) \in \text{succ}(K)$ and (ii) $K' \in L$ s.t. $\text{st}(K, f(\mathbf{x})) \approx \text{st}(K', \mathbf{x})$ and $\langle K', f(v) \rangle \in \text{goal}_q(L)$, then $\langle K, v \rangle \in \text{goal}_q(L)$.

Intuitively, $\text{goal}_q(L)$ tries to construct proofs of q by backward chaining in L ; a proof succeeds, if some pair $\langle K, c \rangle$ is obtained where c is constant symbol. Due to the properties of \mathbb{K}_P , we obtain:

THEOREM 5.16. Let P be an FDNC program and q a ground unary query. Suppose c is the single constant occurring in q . The following two are equivalent:

- (A) $P \models_b q$.

- (B) Some $G \in SM(\mathbf{gp}(P))$ exists such that (i) \mathbb{K}_P is compatible with $\mathbf{st}(G)$, and (ii) some $K \in \mathbb{K}_P$ exists such that $\mathbf{st}(G, c) \approx \mathbf{st}(K, \mathbf{x})$ and $\langle K, c \rangle \in \mathbf{goal}_q(\mathbb{K}_P)$.

By similar arguments as for existential queries, we can see that checking condition (B) is feasible in time single exponential in the size of P and q . Note that computing $\mathbf{goal}_q(\mathbb{K}_P)$ is feasible in time polynomial in the size of \mathbb{K}_P and q , or single exponential in the size of P and q . Once \mathbb{K}_P , $\mathbf{goal}_q(\mathbb{K}_P)$, and $SM(\mathbf{gp}(P))$ are computed, the conditions in (B) can be verified in time polynomial in the combined size of \mathbb{K}_P , $\mathbf{goal}_q(\mathbb{K}_P)$, and $SM(\mathbf{gp}(P))$, each of which is single exponential in the size of P and q . Thus brave entailment of unary ground queries by FDNC programs is in EXPTIME. To establish completeness, recall Theorem 5.11 and item (ii) in Lemma 4.1.

THEOREM 5.17. *For FDC and FDNC programs, brave entailment of unary ground queries (resp., binary ground queries) is EXPTIME-complete.*

5.4 Cautious Entailment of Open Queries

In the previous sections, we presented methods for brave entailment of existentially quantified or ground queries. As shown in Section 4, cautious reasoning can be easily reduced to consistency testing. All these tasks are EXPTIME-complete for FDNC. This section deals with cautious entailment of open queries, which turns out to be harder (under widely adopted beliefs in complexity theory).

Like for other reasoning methods discussed, we base our method on the set \mathbb{K}_P of an FDNC program P . As we have seen, each stable model of P can be constructed by taking a compatible graph and building a tree for each constant. Indeed, if $P \models_c A(t)$ holds, each tree construction starting at the constant of t from knots in \mathbb{K}_P must eventually reach t satisfying A . We introduce the following notion.

DEFINITION 5.18 (CONVERGING SEQUENCE). *Let A be a unary predicate symbol of P . A nonempty sequence $[\frac{L_0}{c}, \frac{L_1}{f_1}, \dots, \frac{L_n}{f_n}]$, where each $L_i \subseteq \mathbb{K}_P$ is nonempty, c is a constant, and f_1, \dots, f_n are function symbols, is called a converging sequence for A (w.r.t. \mathbb{K}_P) if the following hold:*

- (1) for each $K \in L_{j-1}$, where $1 \leq j \leq n$, $f_j(\mathbf{x}) \in \mathbf{succ}(K)$;
- (2) for each $K \in L_{j-1}$, where $1 \leq j \leq n$, and each $K' \in \mathbb{K}_P$, $\mathbf{st}(K, f_j(\mathbf{x})) \approx \mathbf{st}(K', \mathbf{x})$ implies $K' \in L_j$;
- (3) if $K \in L_n$, then $A(\mathbf{x}) \in K$.

Furthermore, we use the following notion for knots that can start models. For a constant c , let $\mathbf{seeds}(c, P)$ be the set of all knots $K \in \mathbb{K}_P$ such that for some $G \in SM(\mathbf{gp}(P))$ we have $\mathbf{st}(K, \mathbf{x}) \approx \mathbf{st}(G, c)$ and \mathbb{K}_P is compatible with $\mathbf{st}(G)$.

PROPOSITION 5.19. *Let P be a consistent FDNC program and let $\lambda x.A(x)$ be an open query. Then $P \models_c \lambda x.A(x)$ iff some converging sequence $s = [\frac{L_0}{c}, \frac{L_1}{f_1}, \dots, \frac{L_n}{f_n}]$ for A exists, where $L_0 = \mathbf{seeds}(c, P)$.*

The proposition above characterizes cautious entailment of open queries in terms of existence of converging sequences. To provide an algorithm, we next show that the length of converging sequences is double exponentially bounded in the size of the initial program.

```

Algorithm openQueries ( $\mathbb{FDNC}$  program  $P$ , open query  $\lambda x.A(x)$ )
Output: true iff there exists  $t$  s.t.  $P \models_c A(t)$ 
if  $P$  is inconsistent then
  return true
end if
Guess some constant  $c$  of  $P$ ;
 $L := \text{seeds}(c, P)$ ;
repeat
  if  $A(\mathbf{x}) \in K$  for each  $K \in L$  then
    return true
  end if
  Guess some  $f \in F$ ;
  if there exists  $K \in L$  such that  $f(\mathbf{x}) \notin \text{succ}(K)$  then
    return false
  else
     $L^{aux} := \{K' \in \mathbb{K}_P \mid \text{st}(K', \mathbf{x}) \approx \text{st}(K, f(\mathbf{x})) \wedge K \in L\}$ ;
     $L := L^{aux}$ ;
     $i := i + 1$ 
  end if
until  $i = |F| \times 2^{|\mathbb{K}_P|} + 1$ 
return false

```

Fig. 5. Non-deterministic procedure for cautious entailment of open queries; \mathbb{K}_P is assumed to be precomputed, F is the set of function symbols of P .

PROPOSITION 5.20. *For every converging sequence $[\frac{L_0}{c}, \frac{L_1}{f_1}, \dots, \frac{L_n}{f_n}]$ for A , there exists a converging sequence $[\frac{L_0}{c}, \frac{L'_1}{f'_1}, \dots, \frac{L'_m}{f'_m}]$ for A such that $m \leq |F| \times 2^{|\mathbb{K}_P|} + 1$, where F is the set of function symbols occurring in P .*

PROOF. There are only $|F| \times 2^{|\mathbb{K}_P|}$ distinct pairs $\frac{L}{f}$ of a function symbol f and a set of knots $L \subseteq \mathbb{K}_P$. Suppose $s = [\frac{L_0}{c}, \frac{L_1}{f_1}, \dots, \frac{L_n}{f_n}]$ is a converging sequence for A and $\frac{L}{f}$ is an element of s that occurs more than once, first at position $0 < k$ and last at position $k < l$. It is easy to verify that $[\frac{L_0}{c}, \frac{L_1}{f_1}, \dots, \frac{L_k}{f_k}, \frac{L_{l+1}}{f_{l+1}}, \dots, \frac{L_n}{f_n}]$ is a converging sequence for A where $\frac{L}{f}$ occurs only once. Note that the first element of the sequence is preserved. It follows that a converging sequence s' for A exists that does not contain duplicates of its elements, while its first element is $\frac{L_0}{c}$. Indeed, s' cannot be longer than $|F| \times 2^{|\mathbb{K}_P|} + 1$, and thus the claim holds. \square

The next theorem follows directly from Proposition 5.19 and Proposition 5.20.

THEOREM 5.21. *Let P be an \mathbb{FDNC} program and let $\lambda x.A(x)$ be an open query. Then $P \models_c \lambda x.A(x)$ iff P is inconsistent or some converging sequence $[\frac{L_0}{c}, \frac{L_1}{f_1}, \dots, \frac{L_n}{f_n}]$ for A exists, where $L_0 = \text{seeds}(c, P)$ and $n \leq |F| \times 2^{|\mathbb{K}_P|} + 1$.*

Based on this theorem, we present in Figure 5.4 an algorithm that decides cautious entailment of open queries by checking the existence of a converging sequence of at most double exponential length. We assume that the set \mathbb{K}_P for the input program P is precomputed. The procedure guesses a sequence of function symbols and verifies the conditions in Definition 5.18. It can be implemented to run in non-deterministic exponential space; indeed, storing the set \mathbb{K}_P and the double exponential counter requires at most exponential space, while the rest of

the constructs require at most linear space. By Savitch's Theorem [1970], we can turn the algorithm into an algorithm using exponential space, which establishes the EXPSPACE-membership. By a generic Turing machine encoding, we show that the problem is EXPSPACE-hard, even for $\mathbb{F}\mathbb{D}$ and $\mathbb{F}\mathbb{N}$ programs (see Appendix).

THEOREM 5.22. *Cautious entailment of open queries in $\mathbb{F}\mathbb{D}$, $\mathbb{F}\mathbb{N}$, $\mathbb{F}\mathbb{N}\mathbb{C}$, $\mathbb{F}\mathbb{D}\mathbb{C}$ and $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ programs is EXPSPACE-complete.*

6. COMPLEXITY OF FRAGMENTS

In this section, we consider the complexity of reasoning in the fragments of $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$. Some reasoning tasks are already covered by the results of Section 5 and the discussion in Section 4, including cautious entailment of existential queries in $\mathbb{F}\mathbb{D}$ (cf. Theorem 5.11 and Lemma 4.1) and of open queries in general (cf. Theorem 5.22).

We first show that in $\mathbb{F}\mathbb{N}$, all reasoning tasks remain as hard as in full $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$. All other reasoning tasks that remain to be considered are at most PSPACE-complete, and in some cases at low levels of the polynomial hierarchy.

6.1 Reasoning in $\mathbb{F}\mathbb{N}$ and $\mathbb{F}\mathbb{N}\mathbb{C}$

We show that the consistency problem for $\mathbb{F}\mathbb{D}\mathbb{C}$ reduces in polynomial time to the consistency problem for $\mathbb{F}\mathbb{N}$. Since the reasoning tasks that we considered (consistency and brave entailment) are EXPTIME-complete for $\mathbb{F}\mathbb{D}\mathbb{C}$ and $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$, the reduction implies that they are all EXPTIME-complete for $\mathbb{F}\mathbb{N}$ and $\mathbb{F}\mathbb{N}\mathbb{C}$.

The plan is as follows. We first construct, given an arbitrary $\mathbb{F}\mathbb{D}\mathbb{C}$ program P , an $\mathbb{F}\mathbb{N}$ program $\text{fr}(P)$ (the *frame* program) whose stable models intuitively coincide with the minimal (forest-shaped) Herbrand interpretations for P . We then structurally transform P into an $\mathbb{F}\mathbb{N}$ program P' such that P is consistent iff the $\mathbb{F}\mathbb{N}$ program $P' \cup \text{fr}(P)$ is consistent.

DEFINITION 6.1 (FRAME PROGRAM $\text{fr}(P)$). *For each predicate Q of P , let \bar{Q} be a fresh predicate symbol of the same arity, Let Dom and S be fresh unary and binary predicate symbols, respectively. Then $\text{fr}(P)$ is the $\mathbb{F}\mathbb{N}$ program with the rules*

$$\begin{array}{ll}
 (F1) & \text{Dom}(c) \leftarrow, \\
 (F2) & S(c, d) \leftarrow, \\
 (F3) & S(x, f(x)) \leftarrow \text{Dom}(x), \\
 (F4) & \text{Dom}(y) \leftarrow S(x, y), \\
 (F5) & A(x) \leftarrow \text{Dom}(x), \text{not } \bar{A}(x) \\
 (F6) & \bar{A}(x) \leftarrow \text{Dom}(x), \text{not } A(x) \\
 (F7) & R(x, y) \leftarrow S(x, y), \text{not } \bar{R}(x, y), \\
 (F8) & \bar{R}(x, y) \leftarrow S(x, y), \text{not } R(x, y),
 \end{array}$$

for each pair c, d of constants of P , each function symbol f of P , and each unary and binary predicate symbol A and R of P , respectively.

PROPOSITION 6.2. *Given an interpretation $I \subseteq \mathcal{H}\mathcal{B}^{\text{fr}(P)}$, $I \in \text{SM}(\text{fr}(P))$ iff I is forest-shaped, and*

- (1) $S(c, d) \in I$, for each pair c, d of constants of P ,
- (2) $\text{Dom}(t) \in I$, for each term $t \hat{\in} I$,
- (3) $S(t, f(t)) \in I$, for each $t \hat{\in} I$ and each function symbol f of P ,
- (4) $|\{A(t), \bar{A}(t)\} \cap I| = 1$, for each $t \hat{\in} I$ and each unary predicate A of P , and
- (5) $S(s, t) \in I$ implies $|\{R(s, t), \bar{R}(s, t)\} \cap I| = 1$, for each pair $s, t \hat{\in} I$ and each binary predicate R of P .

Intuitively, $\text{fr}(P)$ generates a set of forest-shaped interpretations for P . Next we show how to filter out the interpretations that do not satisfy the rules in P . If some interpretation I remains, then P is consistent. Note that such I would not necessarily correspond to a minimal model of P . For technical reasons, we assume that the rules of type (R6) occurring in P are non-disjunctive. It is easy to see that such rules can be eliminated from P in linear time while preserving consistency: replace each rule $R_1(x, f_1(x)) \vee \dots \vee R_n(x, f_n(x)) \leftarrow B_0(x), \dots, B_l(x)$ by rules $A_1(x) \vee \dots \vee A_n(x) \leftarrow B_0(x), \dots, B_l(x)$ and $R_i(x, f_i(x)) \leftarrow A_i(x)$ for each $i \in \{1, \dots, n\}$, where each A_i is a fresh predicate symbol.

DEFINITION 6.3 (TRANSFORMATION). *For an FDC program P as described, we denote by $\text{tf}(P)$ the FN program $\text{fr}(P) \cup P'$, where P' is the FN program obtained from P by replacing each rule*

$$W_1(\vec{t}_1) \vee \dots \vee W_n(\vec{t}_n) \leftarrow Q_1(\vec{v}_1), \dots, Q_m(\vec{v}_m) \in P$$

with a rule

$$C(\vec{t}_1) \leftarrow Q_1(\vec{v}_1), \dots, Q_m(\vec{v}_m), \bar{W}_1(\vec{t}_1), \dots, \bar{W}_n(\vec{t}_n), \text{not } C(\vec{t}_1),$$

where C is a fresh predicate symbol with the arity of \vec{t}_1 , and $n, m \geq 0$.

We note that for simplicity, all rules are rewritten, including non-disjunctive rule and constraints. Indeed, $\text{tf}(P)$ is an FN program; literals $W_i(\vec{t}_i)$ in the head of an initial rule can be shifted to their “complements” $\bar{W}_i(\vec{t}_i)$ in the body without violating the syntax of FN programs. This would not be the case if disjunctive heads were allowed for rules (R6). The following holds (for a proof, see the appendix).

PROPOSITION 6.4. *The program P is consistent iff $\text{tf}(P)$ is consistent.*

We showed how to transform an FDC program into an FN program while preserving consistency. As easily verified, the translation is polynomial in the size of the initial program P (more precisely, quadratic in the size of P due to the facts (F2) of $\text{fr}(P)$; the rest is linear). Therefore, recalling EXPTIME-completeness of consistency testing in FDNC (Theorem 5.11), we conclude.

COROLLARY 6.5. *Checking consistency of FN and FNC programs is EXPTIME-complete.*

The EXPTIME-completeness of consistency checking for FN allows us to obtain similar results for brave query entailment. Since consistency testing is reducible to brave entailment (see Lemma 4.1), and since brave entailment of existential and ground queries is EXPTIME-complete (see Theorems 5.14 and 5.17), we obtain:

COROLLARY 6.6. *For FN and FNC programs, brave entailment of unary (resp., binary, by Lemma 4.1) ground or existential queries is EXPTIME-complete.*

6.2 Reasoning in FC

We show that reasoning in FC is easier than in FDNC: consistency and brave reasoning reduce to PSPACE-completeness. To obtain these results, we cannot exploit the maximal founded set of knots of a program as its size can be exponentially large. Nevertheless, the semantic characterization centering around Theorem 3.13 enables reasoning from FC programs by iterative construction of knots. The following result, which holds for full FDNC, provides a basis for reasoning in FC.

THEOREM 6.7. *Let P be an $\mathbb{F}\text{DNC}$ program. The following two are equivalent:*

- (i) $SM(P) \neq \emptyset$.
- (ii) *There exists some $G \in SM(\mathbf{gp}(P))$ such that, for each constant c of P , a set of knots exists that is founded w.r.t. P and $\{\mathbf{st}(G, c)\}$.*

PROOF. For the “(i) to (ii)” direction, assume that I is a stable model of P . By Theorem 3.13, $\mathit{ffa}(I)$ is a stable model of $\mathbf{gp}(P)$. So let $G = \mathit{ffa}(I)$. By Proposition 3.21, we know that the set of knots $\mathbb{K}(I)$ is founded w.r.t. P and $\mathbf{st}(G)$. Simply take some \subseteq -minimal set L of knots closed under the following rules:

- a) L contains some $K \in \mathbb{K}(I)$ such that $\mathbf{st}(G, c) \approx \mathbf{st}(K, \mathbf{x})$, and
- b) if $K \in L$ and $s \in \mathit{succ}(K)$, then some $K' \in L$ exists such that $\mathbf{st}(K, s) \approx \mathbf{st}(K', \mathbf{x})$.

Indeed, due to foundedness of $\mathbb{K}(I)$, the set L can be constructed and is founded w.r.t. P and $\{\mathbf{st}(G, c)\}$. For the other direction, assume (ii) holds. Let L_c denote a set of knots that is founded w.r.t. P and $\{\mathbf{st}(G, c)\}$. Let C be the set of constants of P . Due to Proposition 3.28, the set $L = \bigcup_{c \in C} L_c$ is a set of knots that is founded w.r.t. P and $\mathbf{st}(G)$. Then Theorem 3.26 proves the claim. \square

The key feature of $\mathbb{F}\mathbb{C}$ is the unique model property, i.e., if an $\mathbb{F}\mathbb{C}$ program has a minimal model, then it is unique. From Theorem 6.7, we know that to decide whether a $\mathbb{F}\mathbb{C}$ program is consistent we can proceed in two steps:

- (1) Check the existence of the single minimal model G of $\mathbf{gp}(P)$. If it exists, then proceed to the next step. Otherwise, P is not consistent.
- (2) Check whether for each constant c of P , a set of knots exists that is founded w.r.t. P and $\{\mathbf{st}(G, c)\}$. If so then P is consistent, otherwise not.

Indeed, G is computable in time polynomial in the size of P . For the second step, notice that the local programs for P also have the unique-model property. This implies a unique set L that is founded w.r.t. P and $\{U\}$, where U is a state.

To decide the second step, in Figure 6 we present a generic non-deterministic procedure *checkCondition*. The procedure takes as input an $\mathbb{F}\text{DNC}$ program P , a state U , and a Boolean function that maps states to Boolean values. In the procedure, the value $b(P)$ is the number of distinct \mathbf{x} -grounded knots over the signature of P . As it was already argued, $b(P) = 2^{n+k \cdot (n+m)}$, where n and m are the numbers of unary and binary predicate symbols of P , respectively, and k is the number of function symbols in P .

Let cond_1 be a Boolean function that maps each state U to *true* if the program $P(U)$ is inconsistent, and to *false* otherwise.

PROPOSITION 6.8. *Let P be an $\mathbb{F}\mathbb{C}$ program, and let U be a state. There exists a set of knots that is founded w.r.t. P and $\{U\}$ iff no run of the procedure $\mathit{checkCondition}(P, U, \mathit{cond}_1)$ returns true.*

PROOF. The “only if” direction is trivial, while for the other direction, we can simply collect all the knots that appeared at any run of the algorithm. It is easy to verify that such a collection is a set that is founded w.r.t. P and $\{U\}$. \square

The algorithm $\mathit{checkCondition}(P, U, \mathit{cond}_1)$ runs in polynomial space. The procedure keeps only a counter that counts up to a single exponential; this requires

```

func checkCondition (program  $P$ , state  $U$ , function cond)
repeat
  if  $cond(U) = true$  then
    return true
  end if;
  Choose  $K \in MM(P(U))$  and  $s \in succ(K)$ ;
  Let  $U$  be a state obtained from  $st(K, s)$  by substituting  $s$  with  $\mathbf{x}$ ;
   $i := i + 1$ 
until  $i = b(P)$ ;
return false

```

Fig. 6. Non-deterministic procedure for PSPACE algorithms

only polynomial space. Note that the procedure at each iteration works only on a single local program that is of polynomial size. This local program has a unique model property and, hence, representing its models requires polynomial space also.

Indeed, to decide the second step, we need to make only a linear number of calls to *checkCondition*. Summing up, both steps to decide consistency of P are feasible in co-NPSPACE w.r.t. to the size of P . By Savitch's Theorem [1970], we know $co\text{-NPSPACE} = \text{PSPACE}$.

THEOREM 6.9. *Deciding whether a given \mathbb{FC} program is consistent is in PSPACE.*

On the other hand, PSPACE-hardness of the problem is shown by a Turing machine simulation (see Lemma A.4 in the appendix). Thus we obtain the following.

THEOREM 6.10. *For \mathbb{FC} programs, checking consistency is PSPACE-complete.*

Since \mathbb{FC} programs have the single-stable model property, brave entailment of existential queries can be easily expressed by constraints that are allowed in \mathbb{FC} .

PROPOSITION 6.11. *Let P be an \mathbb{FC} program. Then $P \models_b \exists x.A(x)$ iff P is consistent and $P \cup \{\leftarrow A(x)\}$ is not consistent.*

The proposition implies that brave entailment of an existential unary query in \mathbb{FC} can be polynomially reduced to consistency checking in \mathbb{FC} . Recalling that the task is PSPACE-hard (Lemma 4.1), we conclude the following.

COROLLARY 6.12. *For \mathbb{FC} programs, brave entailment of unary existential queries is PSPACE-complete. The same holds for binary existential queries (see Lemma 4.1).*

In a similar fashion, we prove PSPACE-completeness for ground queries. The following proposition is helpful.

PROPOSITION 6.13. *Let P be an \mathbb{FC} program and let $A(t)$ be a ground atom such that $t = f_n(\dots f_1(c_0)\dots)$. Let P' result from P by adding the following rules: (a) $C_0(c_0) \leftarrow$, (b) $R(x, f_{i+1}(x)) \leftarrow C_i(x)$, for $0 \leq i < n$, (c) $C_{i+1}(y) \leftarrow C_i(x), R(x, y)$, for $0 \leq i < n$, (d) $D(x) \leftarrow C_n(x), A(x)$, and (e) $\leftarrow D(x)$, where C_0, \dots, C_n, R and D are fresh predicates. Then $P \models_b A(t)$ iff P is consistent and P' is not consistent.*

The proposition implies that brave entailment of a ground unary query in \mathbb{FC} program P can be decided by adding polynomially many rules to P and making two consistency checks, and hence is polynomially reducible to consistency checking in \mathbb{FC} . Since the latter is PSPACE-hard by Lemma 4.1, we have the following result.

THEOREM 6.14. *For \mathbb{FC} programs, brave entailment of unary ground queries is PSPACE-complete. The same holds for binary queries (see Proposition 4.1).*

6.3 Reasoning in \mathbb{F} and \mathbb{FD}

As \mathbb{F} and \mathbb{FD} programs are positive and constraint-free, they are always consistent. We discuss here brave entailment of existential queries together with brave and cautious entailment of ground queries; PSPACE- and EXPTIME-completeness of cautious entailment of existential queries in \mathbb{F} and \mathbb{FD} , respectively, follows from the results for consistency testing in \mathbb{FC} and \mathbb{FDC} (see observation (2) in Section 4).

For a given \mathbb{F} program P , deciding $P \models_b \exists x.A(x)$ is feasible in polynomial space (see Corollary 6.12). On the other hand, the problem is easily seen to be PSPACE-hard; this can be shown by a simple adaptation of the Turing machine simulation for Theorem 6.10 (see Lemma A.5).

THEOREM 6.15. *For \mathbb{F} programs, brave entailment of unary existential queries is PSPACE-complete. The same holds for binary existential queries (see Lemma 4.1).*

For \mathbb{FD} programs, PSPACE-completeness of brave existential queries is not straightforward, since they may have several minimal models and hence the task can not be simply reduced to consistency testing as for \mathbb{F} . The use of constraints leads to \mathbb{FDC} , where consistency testing is already EXPTIME-complete.

The strategy is to use the non-deterministic procedure *checkCondition* from Section 6.2 for consistency testing in \mathbb{FC} . To this end, we observe that the semantic characterization of stable models of \mathbb{FDNC} allows us conclude the following.

THEOREM 6.16. *Let P be a \mathbb{FD} program. The following two are equivalent:*

- (i) $P \models_b \exists x.A(x)$.
- (ii) *There exists $G \in MM(gp(P))$, a constant c of P , and a set of knots L founded w.r.t. P and $\{st(G, c)\}$ such that L contains some knot K with $A(\mathbf{x}) \in K$.*

Let $q = \exists x.A(x)$ be a query and P be an \mathbb{FD} program. The theorem above suggests a method to decide whether $P \models_b q$ holds. The crucial point is to have a procedure for deciding whether for a given state U over P , there exists a set of knots L that is founded w.r.t. P and $\{U\}$ and contains some knot K such that $A(\mathbf{x}) \in K$.

Let $cond_2$ be a Boolean function that maps each world state U to *true* if $A(\mathbf{x}) \in U$, and to *false* otherwise.

PROPOSITION 6.17. *Let U be a state, and P be an \mathbb{FD} program. The following two are equivalent.*

- (i) *Some knot set L founded w.r.t. P and $\{U\}$ and $K \in L$ exist such that $A(\mathbf{x}) \in K$.*
- (ii) *Some execution of $checkCondition(P, U, cond_2)$ returns true.*

PROOF. (i) \Rightarrow (ii): this holds since the size of L is bounded by $b(P)$.

(ii) \Rightarrow (i): consider the sequence of knots that was constructed during the run of the procedure that returned *true*. Since P has no constraints, this sequence can be always augmented to a founded set by computing the missing successors knots. \square

By similar arguments as for consistency checking in \mathbb{FC} , *checkCondition* runs in polynomial space in the input size. Note that traversing the states of constants

that occur in the minimal models of $\text{gp}(P)$ is feasible in polynomial space. Hence, condition (ii) in Theorem 6.16 is testable in polynomial space using a PSPACE oracle; overall, this amounts to polynomial space. As brave entailment of existential queries for \mathbb{FD} programs is PSPACE-hard (see Lemma 4.1), we conclude:

THEOREM 6.18. *For \mathbb{FD} programs, brave entailment of unary existential queries is PSPACE-complete. The same holds for binary existential queries (see Lemma 4.1).*

In contrast to existential queries, brave and cautious reasoning with ground queries is easier in \mathbb{F} and \mathbb{FD} than in \mathbb{FC} and \mathbb{FN} . The methods are based on constructing only relevant parts of stable models to answer a given query. Since \mathbb{F} and \mathbb{FD} do not allow for constraints, we do not need to care about the global consistency of interpretations. By the relevant part of a model, we essentially mean a sequence of knots that is constructed following the path encoded in the term t of a ground query $A(t)$. The following proposition elaborates on that.

PROPOSITION 6.19. *Suppose P be is an \mathbb{FD} program and $A(t)$ a ground atom in which as single constant c occurs. Let $l = \langle s_1, \dots, s_n \rangle$ be the list of subterms of t ordered by increasing term depth, i.e., $s_1 = c$ and $s_n = t$. Then the following hold:*

1. $P \models_b A(t)$ if and only if (\star) there exists some stable model G of $\text{gp}(P)$ and a sequence $\langle K_1, \dots, K_n \rangle$ of stable knots with roots s_1, \dots, s_n , resp., such that:
 - (a) $\text{st}(G, s_1) = \text{st}(K_1, s_1)$,
 - (b) $s_{i+1} \in \text{succ}(K_i)$ and $\text{st}(K_i, s_{i+1}) = \text{st}(K_{i+1}, s_{i+1})$, where $1 \leq i < n$, and
 - (c) $A(s_n) \in K_n$.
2. $P \not\models_c A(t)$ if and only if $(\star\star)$ there exists some model G of $\text{gp}(P)$ and a sequence $\langle K_1, \dots, K_n \rangle$ of knots with roots s_1, \dots, s_n , respectively, such that:
 - (a) $\text{st}(G, s_1) = \text{st}(K_1, s_1)$,
 - (b) $s_{i+1} \in \text{succ}(K_i)$ and $\text{st}(K_i, s_{i+1}) = \text{st}(K_{i+1}, s_{i+1})$, where $1 \leq i < n$,
 - (c) K_i is a model of $P(\text{st}(K_i, s_i))$, where $1 \leq i \leq n$, and
 - (d) $A(s_n) \notin K_n$.

Proposition 6.19 (whose proof is in the appendix) allows us to derive complexity results for ground queries. To ease presentation, the characterization above is via witnessing knot sequences: for brave entailment via a witness for entailment and for cautious entailment via a witness of a counter-model. Note that in (2.c), the knots are not necessarily stable. Stability is not needed, and in fact would hinder finding counter-models in nondeterministic polynomial time (due to stability testing).

Suppose P is an \mathbb{F} program and $A(t)$ a ground query. Since P is a Horn program, the local programs for P have least models computable in polynomial time. Moreover, the least model of $\text{gp}(P)$ is also computable in polynomial time. Hence, $P \models_b A(t)$ can be decided according to Proposition 6.19 by constructing in polynomial time the least model of $\text{gp}(P)$ and the unique sequence of knots. Hence, $P \models_b A(t)$ is in P. On the other hand, since P has the least model, $P \models_b A(t)$ iff $P \models_c A(t)$. Hence, $P \models_c A(t)$ is also in P.

Next suppose that P is an \mathbb{FD} program and $A(t)$ a ground query. As easily seen, testing condition (\star) for P is in Σ_2^P : indeed, guess an interpretation I for $\text{gp}(P)$ and a suitable knot sequence $\langle K_1, \dots, K_n \rangle$ over P 's signature; this results in

a polynomial-size structure. One can then check in polynomial time with an NP oracle whether I is minimal and each knot K_i satisfies the conditions in (\star) .

To decide $P \not\models_c A(t)$, it suffices to verify the condition $(\star\star)$ in Proposition 6.19. Since $(\star\star)$ does not involve minimality of models, it is decidable in NP. Indeed, we may guess an interpretation for $\text{gp}(P)$ and a candidate sequence of knots over the signature of P . Deciding then whether the structure satisfies $(\star\star)$ is feasible in polynomial time. Hence, $P \not\models_c A(t)$ is in NP, while $P \models_c A(t)$ is in co-NP.

It is not difficult to see that the given upper bounds are tight, since they correspond to complexity of brave and cautious reasoning in the propositional case. Simply consider fragments \mathbb{F}_p of \mathbb{F} and \mathbb{FD}_p of \mathbb{FD} that allow only for rules of type (R1), unary facts, and only one constant. Indeed, any propositional Horn (resp. propositional positive disjunctive program) can be rewritten in LOGSPACE into an \mathbb{F}_p (resp. \mathbb{FD}_p) program while preserving the set of minimal models (up to renaming of atoms). This implies that brave/cautious reasoning in propositional Horn and positive disjunctive logic programs are LOGSPACE-reducible to brave/cautious entailment of ground unary queries in \mathbb{F} and \mathbb{FD} programs respectively. Since brave entailment over positive propositional disjunctive programs is Σ_2^P -complete and cautious entailment is co-NP-complete, while both tasks are P-complete for Horn programs (see e.g. [Dantsin et al. 2001]), we obtain completeness results for our formalisms.

THEOREM 6.20. *For \mathbb{FD} programs, brave (resp., cautious) entailment of unary ground queries is Σ_2^P -complete (resp. co-NP-complete). For \mathbb{F} programs, both problems are P-complete. All results extend to binary queries (see Lemma 4.1).*

7. HIGHER-ARITY \mathbb{FDNC}

We present here a decidable extension of \mathbb{FDNC} that supports predicate and function symbols of higher arities, and allows for more succinct and convenient knowledge representation in practice. This will be illustrated by a plain planning scenario (for a more detailed discussion, we refer the interested reader to Appendix B).

As already illustrated by the previous examples, \mathbb{FDNC} supports naturally modeling of possibly infinite evolutions of a set of propositions. Indeed, for a term t , the unary predicates (i.e., the propositions) that hold for it can be used via rules of \mathbb{FDNC} to define the unary predicates that hold for the term $f(t)$, where $f(t)$ can be viewed as a follow-up time point. However, a propositional setting is inconvenient for many action domains, and the use for parameters for more compact representation is needed. They allow to represent actions that, for instance, move an object x from a location l_1 to the location l_2 . Hence, special predicate names for each possible combination of x , l_1 , and l_2 as in a propositional setting can be avoided. (This is, e.g., widely used in [Baral 2002].)

In what follows, we assume that \mathcal{N}_1 and \mathcal{N}_2 are disjoint sets of predicate names of arities at least 1 and 2, respectively.

DEFINITION 7.1 (GLOBAL/LOCAL POSITIONS). *Given an atom $A(t_1, \dots, t_n)$ with $A \in \mathcal{N}_1$ or a term $f(t_1, \dots, t_n)$, its local positions are $1, \dots, n-1$ and its global position is n . Similarly, given an atom $A(t_1, \dots, t_n)$ with $A \in \mathcal{N}_2$, its local positions are $1, \dots, n-2$ and its global positions are $n-1$ and n . An atom $A(\vec{t})$ with $A \in \mathcal{N}_1$ (resp., $A \in \mathcal{N}_2$) is g-unary (resp., g-binary).*

DEFINITION 7.2 (HIGHER-ARITY FDNC PROGRAM). A higher-arity FDNC program is a finite disjunctive logic program whose rules are of the following forms:

$$(R1) \quad \bigvee_{i=1}^k A_i(\vec{v}_i, x) \leftarrow B_0(\vec{u}_0, x), \bigwedge_{j=1}^l B_j^\pm(\vec{u}_j, x)$$

$$(R2) \quad \bigvee_{i=1}^k R_i(\vec{v}_i, x, y) \leftarrow P_0(\vec{u}_0, x, y), \bigwedge_{j=1}^l P_j^\pm(\vec{u}_j, x, y)$$

$$(R3) \quad \bigvee_{i=1}^k R_i(\vec{v}_i, x, f_i(\vec{t}_i, x)) \leftarrow P_0(\vec{u}_0, x, g_0(\vec{w}_0, x)), \\ \bigwedge_{j=1}^l P_j^\pm(\vec{u}_j, x, g_j(\vec{w}_j, x))$$

$$(R4) \quad \bigvee_{i=1}^k A_i(\vec{v}_i, y) \leftarrow P_0(\vec{u}_0, x, y), \bigwedge_{j=1}^l P_j^\pm(\vec{u}_j, x, y), \\ \bigwedge_{j=1}^m B_j^\pm(\vec{w}_j, x), \bigwedge_{j=1}^n C_j^\pm(\vec{t}_j, y)$$

$$(R5) \quad \bigvee_{i=1}^k A_i(\vec{v}_i, f(\vec{v}, x)) \leftarrow P_0(\vec{u}_0, x, f(\vec{v}, x)), \\ \bigwedge_{j=1}^l P_j^\pm(\vec{u}_j, x, f(\vec{v}, x)), \\ \bigwedge_{j=1}^m B_j^\pm(\vec{w}_j, x), \bigwedge_{j=1}^n C_j^\pm(\vec{t}_j, f(\vec{v}, x))$$

$$(R6) \quad \bigvee_{i=1}^k R_i(\vec{v}_i, x, f_i(\vec{t}_i, x)) \leftarrow B_0(\vec{u}_0, x), \bigwedge_{j=1}^l B_j^\pm(\vec{u}_j, x)$$

$$(R7) \quad \bigvee_{i=1}^k A_i(\vec{t}_i, b) \vee \bigvee_{i=1}^l R_i(\vec{v}_i, c, d) \leftarrow \bigwedge_{j=1}^m B_j^\pm(\vec{u}_j, b'), \bigwedge_{j=1}^n P_j^\pm(\vec{w}_j, c', d'),$$

where $k, l, m, n \geq 0$, and

- all A_i and B_j are from \mathcal{N}_1 , and all R_i and P_j are from \mathcal{N}_2 ,
- the tuples \vec{v}, \vec{w} , and all $\vec{v}_i, \vec{t}_i, \vec{u}_j, \vec{w}_j$ are tuples of variables or constants.
- b, c, d, b', c', d' are constants,
- x and y do not occur in local positions of atoms and function symbols, and
- each rule r is safe, i.e., each of its variables occurs in $\text{body}^+(r)$.

The restrictions on the variable interaction allow us to transform higher-arity FDNC programs naturally into ordinary FDNC programs, which enables the usage of reasoning methods from the previous sections. In the following, we present the transformation and a use case of a higher-arity FDNC program.

DEFINITION 7.3 (FDNC REDUCTION). Let P be a higher-arity FDNC program. Let $ld(P)$ be the set of constants occurring in the local positions of atoms in P . Given a set of constants C , every rule r' that results from r by substituting each variable occurring in a local position of some atom of r with some $c \in C$ is a parameter-ground instance of r w.r.t. C ; the set of all such r' is denoted by $gr(r, C)$. The parameter-grounding of P is the program $pgr(P) = \{r' \in gr(r, ld(P)) \mid r \in P\}$.

The FDNC-reduction of P , $red(P)$, results from P by replacing each g -unary atom $A(t_1, \dots, t_n)$ (resp., g -binary atom $R(t_1, \dots, t_n)$) occurring in $pgr(P)$ with an atom $A_{t_1, \dots, t_{n-1}}(t_n)$ (resp., $R_{t_1, \dots, t_{n-2}}(t_{n-1}, t_n)$). Similarly, the FDNC-reduction of an interpretation I for P is defined as $red(I) = \{A_{t_1, \dots, t_{n-1}}(t_n) \mid A(t_1, \dots, t_n) \in I, A \in \mathcal{N}_1\} \cup \{R_{t_1, \dots, t_{n-2}}(t_{n-1}, t_n) \mid R(t_1, \dots, t_n) \in I, R \in \mathcal{N}_2\}$.

The following result is then not difficult to establish.

THEOREM 7.4. *Let P be a higher-arity FDNC program. Then an interpretation I is a stable model of P iff $\text{red}(I)$ is a stable model of $\text{red}(P)$.*

PROOF. We analyze the impact of restricted variable interaction in higher-arity FDNC. As easily verified, the atoms that can be justified (by program rules) in the stable models are of the particular form. Let P be a higher-arity FDNC program, and let $\text{pterm}s(P)$ be the set of *proper* terms defined as the smallest set such that

- a) if $c \in \mathcal{HU}^P$ is a constant, then $c \in \text{pterm}s(P)$;
- b) if $t \in \mathcal{HU}^P$ is a complex term such that (1) in its local positions there are only constants from $\text{ld}(P)$, and (2) in its global positions are the terms from $\text{pterm}s(P)$, then $t \in \text{pterm}s(P)$.

Note that $\text{pterm}s(P)$ is closed under subterms. Let $\text{patom}s(P)$ be the set of all *proper* atoms for P , which are the atoms in \mathcal{HB}^P that have constants from $\text{ld}(P)$ in the local positions and terms from $\text{pterm}s(P)$ in the global positions.

Due to the syntax of higher-arity FDNC, given any (Herbrand) interpretation I of P , a rule $r \in P^I$ contains a non-proper atom iff it contains a non-proper atom in the body. Hence, every $J \in \text{MM}(P^I)$ such that $J \subseteq I$ must satisfy $J \subseteq \text{patom}s(P)$. Let P' result from $\text{Ground}(P)$ by deleting each rule that contains some atom $A \notin \text{patom}s(P)$. Then $\text{MM}(P^I) = \text{MM}(P'^I)$ holds. This implies that $\text{SM}(P) = \text{SM}(P')$. Moreover, only proper atoms can be justified in stable models of P , i.e., $I \subseteq \text{patom}s(P)$ holds for each $I \in \text{SM}(P)$. Trivially, $\text{SM}(P') = \{I \mid \text{red}(I) \in \text{SM}(\text{red}(P'))\}$. On the other hand, it is easily seen that $\text{red}(P') = \text{Ground}(\text{red}(\text{pgr}(P)))$. Since $\text{SM}(\text{Ground}(\text{red}(\text{pgr}(P)))) = \text{SM}(\text{red}(\text{pgr}(P)))$, it follows that $I \in \text{SM}(P)$ iff $\text{red}(I) \in \text{SM}(\text{red}(\text{pgr}(P)))$, as claimed. \square

Since $\text{red}(P)$ is finite, higher-arity FDNC programs inherit decidability from ordinary FDNC programs. However, their complexity is higher (by one exponential) in the general case. This is not surprising, since the parameter-grounding of P is exponential in the size of P .⁵

An exponential blow-up only occurs when arbitrarily many parameters are allowed in rules, i.e., if the number of variables that can occur in local position is unbounded. If the maximal number of variables in local positions is fixed, then the parameter-grounding is polynomial in the size of a higher-arity program, and our complexity results carry over for higher-arity FDNC.

Below is an example of an application of higher-arity FDNC programs to compactly represent the *blocks world* problem. For this purpose, we enhance FDNC programs with “strong” negation $\neg P(\vec{x})$ [Gelfond and Lifschitz 1991], which is expressed in the core language as usual: view $\neg P$ as a fresh predicate symbol and add constraints $\leftarrow P(\vec{x}), \neg P(\vec{x})$.

⁵The hardness results for EXP TIME, CO-NEXP TIME, and CO-NEXP TIME^{NP} corresponding to P, co-NP, and Σ_2^P , respectively, follow from the complexity of ordinary function-free logic programs [Dantsin et al. 2001]; EXPSPACE and 2-EXPSPACE corresponding to PSPACE and EXPSPACE can be obtained by generalizing the given Turing machine encodings. As for 2-EXP TIME corresponding to EXP TIME, one can show EXP TIME-hardness of reasoning in ordinary FDC and FN by adapting the given encoding of PSPACE Turing machines into \mathbb{FC} to *alternating* PSPACE-Turing machines (which capture EXP TIME). With parameters, this can be further lifted to alternating EXPSPACE = 2-EXP TIME.

EXAMPLE 7.5 (ADAPTED FROM [EITER ET AL. 2004]). We assume that initially we have 3 blocks a , b , and c . In the initial state, a and b are on the table (table), while c is on top of a . This is formalized by the following facts:

$$\begin{array}{lll} \text{Block}(a, 0) \leftarrow & \text{On}(a, \text{table}, 0) \leftarrow & \text{Loc}(\text{table}, 0) \leftarrow \\ \text{Block}(b, 0) \leftarrow & \text{On}(b, \text{table}, 0) \leftarrow & \\ \text{Block}(c, 0) \leftarrow & \text{On}(c, a, 0) \leftarrow & \end{array}$$

We need to state the static knowledge about the objects, i.e., the properties of objects that do not change during the execution of actions. We thus state that blocks remain blocks, locations remain locations, and that occupation is determined by having a block on top:

$$\begin{array}{l} \text{Block}(z, y) \leftarrow \text{Block}(z, x), \text{Change}(x, y) \\ \text{Loc}(z, y) \leftarrow \text{Loc}(z, x), \text{Change}(x, y) \\ \text{Loc}(z, x) \leftarrow \text{Block}(z, x) \\ \text{Occupied}(z, x) \leftarrow \text{On}(z_1, z, x), \text{Block}(z, x) \end{array}$$

Next are the effects of action execution. We need to mark the locations that become occupied/unoccupied after moving a block from one location to another. On the other hand, we need to state that the rest of the configuration does not change:

$$\begin{array}{l} \text{On}(y, z, \text{move}(y, z, x)) \leftarrow \text{Block}(y, x), \text{Loc}(z, x), \text{Change}(x, \text{move}(y, z, x)) \\ \neg \text{On}(y, z', \text{move}(y, z, x)) \leftarrow \text{Block}(y, x), \text{Loc}(z', x), \\ \quad \text{Change}(x, \text{move}(y, z, x)), \text{On}(y, z', x), \text{Neq}(z, z') \\ \text{On}(y, z, x') \leftarrow \text{On}(y, z, x), \text{Change}(x, x'), \text{not } \neg \text{On}(y, z, x') \end{array}$$

We use an inequality predicate $\text{Neq}(x, y)$ over parameters, which we axiomatize by adding for each distinct $c_1, c_2 \in \text{ld}(P)$ the fact $\text{Neq}(c_1, c_2) \leftarrow$ to the program.

Next is the executability of an action; only blocks can be moved, and they can only be placed in some location.

$$\text{Change}(x, \text{move}(y, z, x)) \vee \neg \text{Change}(x, \text{move}(y, z, x)) \leftarrow \text{Block}(y, x), \text{Loc}(z, x)$$

The disjunctive rule allows to freely execute the action. Since there might be several blocks that can be moved, the last rule does not force the execution of all applicable actions simultaneously.

The execution of an action can be prohibited by the constraints. In our setting, the block cannot be moved if either the destination is occupied or the block has a block on top of it:

$$\begin{array}{l} \neg \text{Change}(x, \text{move}(y, z, x)) \leftarrow \text{Occupied}(y, x) \\ \neg \text{Change}(x, \text{move}(y, z, x)) \leftarrow \text{Occupied}(z, x) \end{array}$$

We ask whether there exists a sequence of actions that transforms the initial configuration into the one where a is on the table, b is on a and c is on b . This is expressed by the following rule:

$$\text{Plan}(x) \leftarrow \text{On}(c, b, x), \text{On}(b, a, x), \text{On}(a, \text{table}, x)$$

The existence of a plan for the encoded problem can now be decided by the brave query $\exists x. \text{Plan}(x)$ to the constructed higher-arity program. It is easy to verify that there exists a stable model where the following term t satisfies the predicate Plan :

$$t = \text{move}(c, b, \text{move}(b, a, \text{move}(c, \text{table}, 0)))$$

The term t encodes the plan of moving c to the table, b on top of a , and finally c on top of b . The same t is also an answer for the cautious open query $\lambda x. \text{Plan}(x)$ to the program, and encodes a secure plan for the goal.

However, if the initial location of b were not known, i.e., $\text{On}(b, \text{table}, 0) \leftarrow$ is replaced by $\text{On}(b, \text{table}, 0) \vee \text{On}(b, c, 0) \leftarrow$, then the above plan is no longer secure, as the first step is not executable in the case where b is on top of c . Here, the answer

$$t = \text{move}(c, b, \text{move}(b, a, \text{move}(c, \text{table}, \text{move}(b, \text{table}, 0))))$$

to the cautious open query $\lambda x. \text{Plan}(x)$ encodes a secure plan. \square

We finally remark that higher-arity \mathbb{F} DNC programs allow to encode (fragments of) the predicate version of the action language \mathcal{K} , but omit further discussion here.

8. RELATED WORK

Datalog_{nS}. A close relative of \mathbb{F} DNC is *Datalog_{nS}* [Chomicki and Imielinski 1993; Chomicki 1995], which provides an extension of the Datalog language in deductive databases with function symbols more liberal in spirit than in \mathbb{F} DNC programs. The syntax of *Datalog_{nS}* allows for rules in which atoms with complex terms affect atoms with less complex terms, which is not allowed in \mathbb{F} DNC programs. On the other hand, *Datalog_{nS}* features neither of disjunction, negation, and constraints, and thus has to be compared with \mathbb{F} ; modulo minor differences, ordinary and higher-arity \mathbb{F} programs are *Datalog_{nS}* programs.

Chomicki and Imieliński [1993] presented an algebraic approach to compile the least Herbrand models of *Datalog_{nS}* programs (i.e., their single stable models) via homomorphisms into finite structures, on which query answering can be performed. Different representations of these structures, viz. a graph specification and an equational specification that uses a congruence relation, have been described and analyzed; other representation methods for restricted classes of programs in the literature were also discussed. The compilation technique in [Chomicki and Imielinski 1993] does not extend to \mathbb{F} DNC programs, which can have multiple (even infinitely many) stable models. The techniques of knots, which constitute building blocks of stable models, handles multiplicity of models by knot sharing, such that every stable model can be readily assembled from knots.

Notably, ordinary and higher-arity \mathbb{F} have lower complexity than *Datalog_{nS}*, at least regarding data complexity (which was considered in [Chomicki and Imielinski 1993]). As reported there, cautious entailment of ground queries in *Datalog_{nS}* is EXPTIME-complete with respect to data complexity, i.e., w.r.t. the size of the set of facts in the program. On the other hand, cautious entailment of ground queries from \mathbb{F} programs (which coincides with brave entailment) is feasible in polynomial time; the same holds for higher-arity \mathbb{F} programs when the number of parameters in each rule is bounded by a constant, since then the parameter grounding $\text{pgr}(P)$ and the \mathbb{F} DNC-reduct of P have polynomial size; thus, the ground query can be answered in polynomial time when the rules are fixed. This continues to hold when facts added to P may also involve function symbols (in global positions only): complex terms in facts can be compiled away in polynomial time (e.g., by partial instantiation and introducing fresh predicate and constant symbols for ground terms). Hence, w.r.t.

data complexity, our \mathbb{F} programs constitute a meaningful, tractable fragment of $Datalog_{nS}$. In [Chomicki 1995], different evaluation strategies for query answering from $Datalog_{nS}$ programs have been considered; by their relationship to \mathbb{F} programs, they can be applied to the latter as well.

Finitely recursive and finitary programs. Our class \mathbb{FN} , which results from \mathbb{FDNC} by disallowing constraints and disjunction, is in essence (modulo elimination of rules (R2) and (R4)) a decidable subclass of the *Finitely Recursive Programs (FRPs)* [Bonatti 2004; Baselice et al. 2007]. FRPs are normal logic programs P with function symbols such that in $\text{Ground}(P)$ each atom depends only on finitely many atoms. In this formalism, inconsistency checking is r.e.-complete and brave ground entailment is co-r.e.-complete in general [Baselice et al. 2007]. 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)* [Bonatti 2004], 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 all other fragments of \mathbb{FDNC} allows effective recognition of their programs. Recognition of FRPs and FPs, instead, suffer from undecidability.

Omega-restricted logic programs. For logic programs with negation under stable model semantics, ω -restricted logic programs have been presented in [Syrjänen 2001]. These are normal logic programs with function symbols of arbitrary arities and an unbounded number of variables, but have restricted syntax to ensure the finite-model property, i.e., that finite answer sets always exist. The restriction is a generalization of classical stratification based on the existence of an acyclic ordering of the atom dependencies, which adds a special ω -stratum that holds all unstratifiable predicates of the logic program. In contrast, our \mathbb{FDNC} programs do not exclude cyclic dependencies, and they lack the finite model property. Furthermore, \mathbb{FDNC} programs have lower computational complexity. While consistency testing in general ω -restricted programs is 2-NEXPTIME-complete, the test can be done in EXPTIME for ordinary and in 2-EXPTIME for higher-arity \mathbb{FDNC} programs.

Local extended conceptual logic programs. Another formalism related to \mathbb{FDNC} -programs are *Local Extended Conceptual Logic Programs (LECLPs)* [Heymans et al. 2005] which evolved from [Heymans 2006]. Such programs are function-free but have answer sets over *open domains*, i.e., of the grounding of P with an arbitrary superset of the constants in P . LECLPs are syntactically restricted to ensure the forest-shape model property of answer sets. Deciding consistency of an LECLP P is feasible in 3-NEXPTIME, as one can ground P with double exponentially many constants in the size of P , and then use a standard ASP solver. For \mathbb{FDNC} , deciding the consistency is EXPTIME-complete and thus less complex.

Comparing the expressiveness of LECLPs and \mathbb{FDNC} is intricate due to 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

deviate from the general intuition behind the minimal model semantics of logic programs. So-called *free rules* of the form $p(x) \vee \text{not } p(x) \leftarrow$; allow to unfoundedly add atoms for p in an answer set. FDNC , instead, has no free rules, and each atom in a stable model of P must be justified from the facts of P .

Reductions of Description Logics to ASP. Reductions of Description Logics to ASP have been considered e.g. in [Alsaç and Baral 2001; Baral 2002; Swift 2004; Hustadt et al. 2004; Heymans and Vermeir 2003; Heymans et al. 2005]. Alsaç and Baral [2001; 2002] gave a reduction of \mathcal{ALCQI} to normal function-free logic programs (i.e., Datalog with stable negation), which was geared towards the Herbrand domain of a knowledge base; by adding rules to generate inductively terms with a function symbol, they extended it to infinite domains. Their reduction is, in a sense, less constructive than the one given here and others, where function symbols are used to handle existential quantifiers by Skolemization. Swift [2004] reported a reduction of deciding satisfiability of \mathcal{ALCQI} concepts to Datalog with stable negation, which exploits the finite model property of this problem. Heymans et al. [2003; 2005] reduced \mathcal{SHIQ} (which subsumes \mathcal{ALCQI}) to their Conceptual Logic Programs and extensions; however, they used answer sets over open domains.

Most relevant for the present paper is Hustadt et al.'s [2004]. They reduced reasoning in \mathcal{SHIQ} to the evaluation of a positive disjunctive Datalog program. The program is generated in three steps. First, the knowledge base is translated into first-order logic in a standard way. After that, resolution and superposition techniques are applied to saturate a clausal form of the transformation. Finally, function symbols are removed using new constant symbols.

The reduction of \mathcal{ALC} to FDC in Section 5.2 is, in essence, similar to others and a close relative of the one of Hustadt et al. described above. The main differences to the latter are with respect to their second step, where our method uses knots for compilation, and that our method aims at model building while the one in [Hustadt et al. 2004] is geared towards query answering. Notably, the disjunctive Datalog program constructed in [Hustadt et al. 2004] is generally exponential in the size of the initial DL knowledge base (but is evaluable in co-NP), while the FDC program is polynomial (but may need exponential time for evaluation).

Furthermore, the reduction contributes in two respects. First, the knowledge base is rewritten (very efficiently) on the DL syntax side into a normal form, rather than on the first-order logic side after the mapping. Second, a transformation into FDC opens the possibility to use any dedicated evaluation algorithm for such programs, beyond a specific method (like the one in this paper).

Reasoning about Actions and Planning. As already discussed in the previous sections, the use of non-monotonic logic programs under answer set semantics as a tool for solving problems in reasoning about actions has been considered in many papers, including [Dimopoulos et al. 1997; Lifschitz 1999; Baral 2002; Eiter et al. 2004; Tu et al. 2007; Son et al. 2006; Son et al. 2005; Morales et al. 2007]. The work presented here adds to these other works by providing an underpinning of the computational properties of non-monotonic logic programs with functions symbols that naturally emerge in this context, and, importantly, capture indefinitely long action sequences. They help in assessing the complexity of particular problems and

may be useful to show that tractability can be achieved in some cases. Furthermore, our results provide algorithms to solve problems expressed in the language.

9. CONCLUSION

In line with efforts to pave the way for effective Answer Set Programming engines with function symbols [Bonatti 2004; Baselice et al. 2007], we presented $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ programs as a decidable class of disjunctive logic programs with function symbols under stable model semantics.

$\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ and its subclasses are a powerful tool for knowledge representation and reasoning for some applications involving infinite processes and objects, like evolving action domains. They are, by their intrinsic complexity, the proper fragment of logic programs to capture secure (alias conformant) planning in declarative action languages with a transition-based semantics like \mathcal{K} , \mathcal{C} , and similar languages, which is an EXPSPACE -complete problem.

Notably, $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ programs can have infinitely many and infinitely large stable models. To finitely represent those models, we introduced a technique that allows to reconstruct stable models as forests from so called *knots* from maximal founded set of knots. The finite representation technique allowed us to define an elegant decision procedure for brave reasoning in $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$, and may also be exploited for offline knowledge compilation to speed up online reasoning and model building, by precomputing and storing the knots of a program.

Furthermore, we have characterized the complexity of reasoning in $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ programs, which is lower than in the most popular of the recent approaches to enhance Answer Set Programming with functions symbols. $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ and its subclasses provide *effective syntax* for expressing problems in PSPACE , EXPTIME , and EXPSPACE using logic programs with function symbols.

There are several avenues for future research. One is to generalize the syntax of $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$, while keeping decidability and benign semantic properties. An interesting such extension is to allow rules of the form $R(y, x) \leftarrow R(x, y)$. The ability to define inverse relations, which are common in expressive DLs and related modal logics, would allow for the bi-directional flow of information between elements in the domain; the resulting programs are, as seen by a reduction to monadic second-order logic over trees \mathcal{SkS} , still decidable. While the forest-shaped model property would remain, such relations are problematic due to the stable model semantics. In presence of inverse relations, testing the minimality of interpretation for a program becomes more involved. Intuitively, the justification of atoms in an interpretation can no longer be verified by only considering the structurally less complex atoms. To deal with these issues, we plan to generalize the knot-based technique for the finite representation of stable models.

$\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ and, in particular, the finite representation of stable models is a promising basis of developing algorithms for answering more complex queries than those considered in this paper. Since $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ easily captures some basic DLs, the algorithms developed for $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ may be applicable also in other domains. In general, query answering algorithms need to construct a set of models in order to answer the query. The algorithms using the maximal founded set of knots as an input, would be relieved from computationally expensive model building since the relevant part

of the model can be built using knots without the need to ensure the consistency.

Finally, implementation of $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ programs is a subject of future work. Since stable knots are defined as stable models of local programs (which are finite propositional disjunctive logic programs), the implementation will certainly export parts of reasoning to one of the highly optimized answer set solvers currently available. In particular, recent extensions of the DLV system like DLVHEX, which implements hex programs [Eiter et al. 2005] that feature external function calls (by which limited Skolemization could be simulated), may be attractive for this.

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library by visiting the following URL: <http://www.acm.org/pubs/citations/journals/tocl/2099-9-9/p1-URL>.

Acknowledgments

We are grateful to the reviewers of this and the preliminary LPAR 2007 paper for their useful comments and suggestions for improvements, as well as to Piero Bonatti, Nicola Leone, Stijn Heymans and many further colleagues.

REFERENCES

- ALSAÇ, G. AND BARAL, C. 2001. Reasoning in description logics using declarative logic programming. Tech. rep., Dep. Computer Science and Engineering, Arizona State University.
- ANDREKA, H. AND NEMETI, I. 1978. The generalised completeness of Horn predicate logics as programming language. *Acta Cybernetica* 4, 1, 3–10.
- ASPARAGUS. Homepage (since 2005). <http://asparagus.cs.uni-potsdam.de/>.
- BAADER, F., BRANDT, S., AND LUTZ, C. 2005. Pushing the EL envelope. See Kaelbling and Saffiotti [2005], 364–369.
- BAADER, F., CALVANESE, D., MCGUINNESS, D., NARDI, D., AND PATEL-SCHNEIDER, P. F., Eds. 2003. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press.
- BARAL, C. 2002. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.
- BASELICE, S., BONATTI, P. A., AND CRISCUOLO, G. 2007. On finitely recursive programs. In *Proc. 23rd Int'l Conference on Logic Programming (ICLP 2007)*. LNCS 4670. Springer, 89–103.
- BONATTI, P. A. 2004. Reasoning with infinite stable models. *Artificial Intelligence* 156, 1, 75–111.
- CADOLI, M. AND DONINI, F. 1997. A survey on knowledge compilation. *AI Comm.* 10, 3-4, 137–150.
- CHOMICKI, J. 1995. Depth-bounded bottom-up evaluation of logic programs. *Journal of Logic Programming* 25, 1, 1–31.
- CHOMICKI, J. AND IMIELINSKI, T. 1993. Finite representation of infinite query answers. *ACM Transactions on Database Systems* 18, 2, 181–223.
- DANTSIN, E., EITER, T., GOTTLÖB, G., AND VORONKOV, A. 2001. Complexity and expressive power of logic programming. *ACM Computing Surveys* 33, 3, 374–425.
- DARWICHE, A. AND MARQUIS, P. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research* 17, 229–264.
- DIMOPOULOS, Y., NEBEL, B., AND KOEHLER, J. 1997. Encoding planning problems in nonmonotonic logic programs. In *Proc. European Conference on Planning 1997 (ECP-97)*. LNCS 1348. Springer, 169–181.
- DIX, J., FURBACH, U., AND NERODE, A., Eds. 1997. *Proc. 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'97)*. LNCS 1265. Springer.

- EITER, T., FABER, W., LEONE, N., PFEIFER, G., AND POLLERES, A. 2003. A logic programming approach to knowledge-state planning, II: The dlv^K system. *Artif. Intell.* 144, 1-2, 157–211.
- EITER, T., FABER, W., LEONE, N., PFEIFER, G., AND POLLERES, A. 2004. A logic programming approach to knowledge-state planning: Semantics and complexity. *ACM Transactions on Computational Logic* 5, 2, 206–263.
- EITER, T. AND GOTTLÖB, G. 1997. Expressiveness of stable model semantics for disjunctive logic programs with functions. *Journal of Logic Programming* 33, 2, 167–178.
- EITER, T., GOTTLÖB, G., AND MANNILA, H. 1997. Disjunctive datalog. *ACM Transactions on Database Systems* 22, 3, 364–418.
- EITER, T., IANNI, G., SCHINDLAUER, R., AND TOMPITS, H. 2005. A uniform integration of higher-order reasoning and external evaluations in answer set programming. See Kaelbling and Saffiotti [2005], 90–96.
- EITER, T., LEONE, N., MATEIS, C., PFEIFER, G., AND SCARCELLO, F. 1997. A deductive system for non-monotonic reasoning. See Dix et al. [1997], 364–375.
- GEBSEER, M., KAUFMANN, B., NEUMANN, A., AND SCHAUB, T. 2007. *Clasp* : A conflict-driven answer set solver. In *Proc. LPNMR-07*, C. Baral, G. Brewka, and J. S. Schlipf, Eds. Lecture Notes in Computer Science, vol. 4483. Springer, 260–265.
- GELFOND, M. AND LIFSCHITZ, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9, 3/4, 365–386.
- GELFOND, M. AND LIFSCHITZ, V. 1992. Representing actions in extended logic programming. In *Proc. Joint Int'l Conf. and Symp. on Logic Programming (JICSLP-92)*. MIT Press, 559–573.
- GIUNCHIGLIA, E. AND LIFSCHITZ, V. 1998. An action language based on causal explanation: Preliminary report. In *Proc. 15th National Conf. Artificial Intelligence (AAAI-98)*. 623–630.
- HANKS, S. AND MCDERMOTT, D. V. 1987. Nonmonotonic logic and temporal projection. *Artificial Intelligence* 33, 3, 379–412.
- HASLUM, P. AND JONSSON, P. 1999. Some results on the complexity of planning with incomplete information. In *Proc. 5th European Conf. Planning (ECP-99)*. LNCS 1809, 308–318.
- HERBRAND, J. 1971. *Logical Writings*. Harvard University Press. Edited by Warren D. Goldfarb.
- HEYMANS, S. 2006. Decidable open answer set programming. Ph.D. thesis, Theoretical Computer Science Lab, Department of Computer Science, Vrije Universiteit Brussel.
- HEYMANS, S., NIEUWENBORGH, D. V., AND VERMEIR, D. 2005. Nonmonotonic ontological and rule-based reasoning with extended conceptual logic programs. In *Proc. 2nd European Semantic Web Conference (ESWC-05)*, LNCS 3532. Springer, 392–407.
- HEYMANS, S. AND VERMEIR, D. 2003. Integrating semantic web reasoning and answer set programming. In *Proc. Workshop on Answer Set Programming (ASP-2003)*, M. de Vos and A. Proveti, Eds. CEUR Workshop Proc., vol. 78. CEUR-WS.org, 194–208.
- HUSTADT, U., MOTIK, B., AND SATTTLER, U. 2004. Reducing SHIQ-description logic to disjunctive datalog programs. In *Proceedings KR-2004*. AAAI Press, 152–162.
- HUSTADT, U., SCHMIDT, R. A., AND GEORGIEVA, L. 2004. A survey of decidable first-order fragments and description logics. *J. Relational Methods in Computer Science* 1, 251–276.
- ITAI, A. AND MAKOWSKY, J. A. 1987. Unification as a complexity measure for logic programming. *Journal of Logic Programming* 4, 105–117.
- KAELBLING, L. P. AND SAFFIOTTI, A., Eds. 2005. *IJCAI-05, Proc. 19th International Joint Conference on Artificial Intelligence*. Professional Book Center.
- LEVESQUE, H. J., PIRRI, F., AND REITER, R. 1998. Foundations for the situation calculus. *Electronic Transactions on Artificial Intelligence* 2, 159–178.
- LIFSCHITZ, V. 1999. Answer set planning. In *Proc. 16th International Conference on Logic Programming (ICLP-99)*, D. D. Schreye, Ed. The MIT Press, 23–37.
- LIFSCHITZ, V. 2002. Answer set programming and plan generation. *Artif. Intell.* 138, 39–54.
- LIFSCHITZ, V. AND TURNER, H. 1994. Splitting a logic program. In *Proc. 11th International Conference on Logic Programming (ICLP-94)*, P. V. Hentenryck, Ed. The MIT Press, 23–37.
- MAREK, V. W. AND REMMEL, J. B. 2003. On the expressibility of stable logic programming. *Theory and Practice of Logic Programming* 3, 4-5, 551–567.

- MAREK, V. W. AND TRUSZCZYŃSKI, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm – A 25-Year Perspective*, Springer, 375–398.
- MAREK, W., NERODE, A., AND REMMEL, J. 1992. How complicated is the set of stable models of a recursive logic program? *Annals of Pure and Applied Logic* 56, 119–135.
- MAREK, W., NERODE, A., AND REMMEL, J. 1994. The stable models of a predicate logic program. *Journal of Logic Programming* 21, 3, 129–153.
- MORALES, A. R., TU, P. H., AND SON, T. C. 2007. An extension to conformant planning using logic programming. In *Proc. 20th Int'l Joint Conference on Artificial Intelligence (IJCAI-07)*. AAAI Press/IJCAI, 1991–1996.
- MOTIK, B., HORROCKS, I., AND SATTLER, U. 2007. Bridging the gap between OWL and relational databases. In *Proc. 16th Int'l Conf. World Wide Web (WWW-07)*. ACM, 807–816.
- NIEMELÄ, I. 1999. Logic programming with stable model semantics as constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence* 25, 3–4, 241–273.
- NIEMELÄ, I. AND SIMONS, P. 1997. Smodels - an implementation of the stable model and well-founded semantics for normal lp. See Dix et al. [1997], 421–430.
- SAVITCH, W. J. 1970. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences* 4, 2, 177–192. ISSN 1439-2275.
- SCHILD, K. 1991. A correspondence theory for terminological logics: Preliminary report. In *Proc. 12th Int'l Joint Conf. on Artificial Intelligence (IJCAI-91)*. Morgan Kaufmann, 466–471.
- SON, T. C., BARAL, C., TRAN, N., AND MCILRAITH, S. A. 2006. Domain-dependent knowledge in answer set planning. *ACM Transactions on Computational Logic* 7, 4, 613–657.
- SON, T. C., TU, P. H., GELFOND, M., AND MORALES, A. R. 2005. Conformant planning for domains with constraints—a new approach. In *Proc. 20th National Conf. Artificial Intelligence (AAAI-05)*. AAAI Press/MIT Press, 1211–1216.
- SWIFT, T. 2004. Deduction in ontologies via ASP. In *Proc. 7th Int'l Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR-04)*. LNCS/LNAI 2923, Springer, 275–288.
- SYRJÄNEN, T. 2001. Omega-restricted logic programs. In *Proc. 6th Int'l Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR-01)*. LNCS 2173, Springer, 267–279.
- TU, P. H., SON, T. C., AND BARAL, C. 2007. Reasoning and planning with sensing actions, incomplete information, and static causal laws using answer set programming. *Theory and Practice of Logic Programming* 7, 4, 377–450.
- WOLTRAN, S. 2005. Answer set programming: Model applications and proofs-of-concept. Tech. Rep. WP5, Working Group on Answer Set Programming (WASP, IST-FET-2001-37004). July. Available at <http://www.kr.tuwien.ac.at/projects/WASP/report.html>.

Received January 2008; revised December 2008; accepted February 2009