

THIS DOCUMENT IS THE ONLINE-ONLY APPENDIX TO:

FDNC: Decidable Nonmonotonic Disjunctive Logic Programs with Function Symbols

THOMAS EITER and MANTAS ŠIMKUS
Technische Universität Wien

ACM Transactions on Computational Logic, Vol. 9, No. 9, June 2009, Pages 1–45.

A. PROOFS AND CONSTRUCTIONS

A.1 Auxiliary Lemma

PROOF OF LEMMA 4.1. The statement (i) follows directly from the fact that in the basic fragment \mathbb{F} we can state unary and binary facts. Indeed, P is consistent iff $P \cup \{Q(c) \leftarrow\} \models_b \exists x.Q(x)$, where Q and c are fresh symbols not occurring in P . Hence, whenever a fragment allows for unary facts, the consistency problem in that fragment can be reduced in logarithmic space to brave entailment of existential unary queries in the same fragment. The same can be shown for binary existential queries, and also for ground queries.

It is easy to see that the statement (ii) holds for existential queries. Indeed, for an arbitrary logic program P , the following hold:

- 1) $P \models_b \exists x, y.R(x, y)$ iff $P \cup \{Q(x) \leftarrow R(x, y)\} \models_b \exists x.Q(x)$, and
- 2) $P \models_b \exists x.A(x)$ iff $P \cup \{W(x, f(x)) \leftarrow A(x)\} \models_b \exists x, y.W(x, y)$,

where Q , W , and f are fresh symbols not occurring in P . This defines a logarithmic space reduction from brave entailment of binary existential queries to unary ones, and vice versa. Since even in the basic \mathbb{F} fragment the syntax allows to add the necessary rule, the claim follows.

As in the case above, by utilizing additional rules, brave entailment of binary ground queries can be reduced in logarithmic space to brave entailment of unary ground queries, and vice versa. Hence, the statement (ii) also holds for ground queries. We state the properties that allow for reduction. Let q be a binary ground atom, and let P be an FDNC program. Due to the forest-shape model property, if q is not of the form (a) $R(c, d)$ or (b) $R(t, f(t))$, where c, d are constants, then $P \not\models_b q$. Therefore, without loss of generality, we can assume that binary queries over FDNC programs are of the form (a) or (b). The reduction then follows from the following properties:

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

a) $P \models_b R(c, d)$ iff

$$P \cup \{R'(c, d) \leftarrow; R''(x, y) \leftarrow R(x, y), R'(x, y); Q(y) \leftarrow R''(x, y)\} \models_b Q(d),$$

b) $P \models_b R(t, f(t))$ iff $P \cup \{Q(y) \leftarrow R(x, y)\} \models_b Q(f(t))$,

c) $P \models_b A(v)$ iff $P \cup \{R'(x, f(x)) \leftarrow A(x)\} \models_b R'(v, f(v))$,

where Q, R', R'' , and f are fresh symbols not occurring in P and v, t are ground.

For the statement (iii), it is easy to see that cautious entailment of unary open queries can be reduced in linear time to cautious entailment of binary open queries. Indeed, $P \models_c \lambda x.A(x)$ with the answer $x = t$ iff $P \cup \{R(x, f(x)) \leftarrow A(x)\} \models_c \lambda x, y.R(x, y)$ with the answer $x = t, y = f(t)$, where R and f are fresh symbols not occurring in P . For the reduction in the other direction, consider a $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ program P and a query $\lambda x, y.R(x, y)$. We define the program P' obtained from P by adding

(a) for each pair c, d of constants of P , the rules

- $R'_{c,d}(c, d) \leftarrow$,
- $R_{c,d}(x, y) \leftarrow R'_{c,d}(x, y), R(x, y)$, and
- $A_{c,d}(y) \leftarrow R_{c,d}(x, y)$,

where $R'_{c,d}, R_{c,d}$ and $A_{c,d}$ are fresh symbols, and

(b) for each function symbol f of P , the rule $A_f(f(y)) \leftarrow R(x, f(x))$, where A_f is a fresh symbol.

It is easy to verify that $P \models_c \lambda x, y.R(x, y)$ iff at least one of the following holds:

1. for some pair c, d of constants of P , $P \models_c \lambda x.A_{c,d}(x)$, or
2. for some function symbol f of P , $P \models_c \lambda x.A_f(x)$.

where each of the predicate symbols in the heads is a fresh symbol. By this construction, cautious entailment of a binary open query can be decided by polynomially many cautious entailment problems of unary open queries that are constructible in polynomial time. Hence statement (iii) holds. \square

A.2 Normalization of \mathcal{ALC} KBs

We show how to transform in linear time an arbitrary \mathcal{ALC} KB \mathcal{K}_1 into a KB \mathcal{K}_2 such that \mathcal{K}_2 is in normal form, is safe, and \mathcal{K}_1 is satisfiable iff \mathcal{K}_2 is satisfiable (i.e., \mathcal{K}_1 and \mathcal{K}_2 are equi-satisfiable). For technical reasons, we assume that \mathcal{ALC} KBs contain only concepts that are in *negation normal form*, i.e., negation may occur only in front of atomic concepts. It is well known that an arbitrary \mathcal{ALC} concept can be transformed in linear time into an equivalent concept in negation normal form. We start with the transformation into normal form and then move to safety of KBs.

Given an arbitrary \mathcal{ALC} KB \mathcal{K} , an equi-satisfiable KB \mathcal{K}' in normal form can be obtained by exhaustive rewriting of axioms in \mathcal{K} using the rules in Table III. The rewriting is performed in 4 phases. It is easy to verify that the transformation is terminating, preserves the consistency, and after the exhaustive rewriting in the final Phase 4 yields a KB in normal form.

We analyze the computational complexity of the rewriting in each of the phases. Following the standard assumption in Description Logics, we assume that each of the atomic concepts in \mathbf{C} is of constant size, i.e., the length of the binary string

Ph.1	$D \oplus \hat{C} \sqsubseteq E \rightsquigarrow D \oplus A \sqsubseteq E, \hat{C} \sqsubseteq A$
	$D \sqsubseteq E \oplus \hat{C} \rightsquigarrow D \sqsubseteq E \oplus A, A \sqsubseteq \hat{C}$
	$\mathbb{Q}R.\hat{C} \sqsubseteq E \rightsquigarrow \hat{C} \sqsubseteq A, \mathbb{Q}R.A \sqsubseteq E$
	$D \sqsubseteq \mathbb{Q}R.\hat{C} \rightsquigarrow D \sqsubseteq \mathbb{Q}R.A, A \sqsubseteq \hat{C}$
Ph.2	$\hat{C} \sqsubseteq \hat{D} \rightsquigarrow \hat{C} \sqsubseteq A, A \sqsubseteq \hat{D}$
	$C \sqcup D \sqsubseteq B \rightsquigarrow C \sqsubseteq B, D \sqsubseteq B$
	$B \sqsubseteq C \sqcap D \rightsquigarrow B \sqsubseteq C, B \sqsubseteq D$
Ph.3	$\mathbb{Q}R.B \sqsubseteq D \rightsquigarrow \top \sqsubseteq A \sqcup D, A \sqsubseteq \mathbb{Q}^-R.A', A' \sqcap B \sqsubseteq \perp$
Ph.4	$C \sqsubseteq D \sqcup \neg E \rightsquigarrow C \sqcap E \sqsubseteq D$
	$C \sqcap \neg D \sqsubseteq E \rightsquigarrow C \sqsubseteq D \sqcup E$
	$\perp \sqcap D \sqsubseteq E \rightsquigarrow \emptyset$
	$D \sqsubseteq E \sqcup \top \rightsquigarrow \emptyset$
	$\top \sqcap D \sqsubseteq E \rightsquigarrow D \sqsubseteq E$
	$D \sqsubseteq E \sqcup \perp \rightsquigarrow D \sqsubseteq E$

where $\oplus \in \{\sqcap, \sqcup\}$, $\mathbb{Q} \in \{\forall, \exists\}$, concepts \hat{C}, \hat{D} are not literal concepts, A, A' are fresh concepts, B is atomic, the rest are arbitrary.

Table III. Rules for Rewriting into Normal Form

representing an atomic concept does not depend on the particular knowledge base. The size $|\mathcal{K}|$ of a knowledge base \mathcal{K} amounts then to the number of symbols in the string representing the axioms of \mathcal{K} . Without loss of generality, we assume that \mathcal{K} contains only one axiom α (note that, in general, each axiom in a knowledge base can be rewritten independently).

It is easy to see that in Phase 1 the number of rewritings is bounded by $c + q$, where c and q respectively denote the number of binary connectives, and quantifiers (“ \forall ” or “ \exists ”) occurring in \mathcal{K} . Since each application of a rule removes an axiom and adds two axioms, the number of axioms resulting by rewriting α is bounded by $c + q$. Since the application of a rewrite rule to an axiom yields two axioms whose combined size increases by some fixed constant not depending on the size of the KB (due to the assumption on the size of atomic concepts), the rewriting in Phase 1 is feasible in linear time in the size of the initial KB.

Phase 2 is feasible in linear time in the size of the knowledge base obtained in Phase 1. Indeed, only linearly many rule applications can occur and each of the rewriting causes a constant overhead in the representation of new axioms.

Phase 3 that deals with the elimination of quantifier in the antecedent of an axiom is clearly linear in the size of the KB obtained in Phase 3.

In Phase 4 the number of rewrite steps is bounded by the number of negation symbols and occurrences of \top and \perp in the knowledge base resulting from Phase 3, i.e., it is clearly linear.

Since each phase requires at most linear time in the size of the input, we conclude that normalizing a KB \mathcal{K} is feasible in linear in the size of \mathcal{K} .

We now show that each \mathcal{ALC} KB in normal form can be transformed in linear time into a safe KB in normal form while preserving the consistency. For a given KB

\mathcal{K} we can construct the safe knowledge base \mathcal{K}' by modifying \mathcal{K} in the following way:

- for each individual name i occurring in \mathcal{K} , adding the assertion $Dom(i)$ to \mathcal{K} ,
- for each role R of \mathcal{K} , adding $Dom \sqsubseteq \forall R.Dom$ to \mathcal{K} , and
- replacing each axiom $\top \sqsubseteq D \in \mathcal{K}$ of type (T3), by $Dom \sqsubseteq D$,

where Dom is a fresh concept name not occurring in \mathcal{K} . Indeed, \mathcal{K}' is safe and in normal form by construction. It is easy to verify that \mathcal{K} is consistent iff \mathcal{K}' is consistent. Indeed, if \mathcal{I} is a first-order interpretation that is a model of $\Theta(\mathcal{K})$, then we can extend \mathcal{I} to be a model of $\Theta(\mathcal{K}')$ by extending \mathcal{I} to interpret Dom as the whole domain of \mathcal{I} . For the other direction, suppose \mathcal{K}' is consistent. Since \mathcal{ALC} has the forest-shaped model property (cf. [Baader et al. 2003]), due to the construction, there exists a model \mathcal{I} of $\Theta(\mathcal{K}')$ where every domain element satisfies Dom . Then, trivially, \mathcal{I} is a model of $\Theta(\mathcal{K})$. The construction of \mathcal{K}' is clearly linear in the size of \mathcal{K} .

A.3 Brave Entailment in \mathbb{FDNC}

PROOF OF THEOREM 5.13. Suppose (A) holds, i.e., there exists some $I \in SM(P)$ such that $A(t) \in I$, for some term t . By Theorem 3.13, $ffa(I) \in SM(\mathbb{gp}(P))$. Let $G = ffa(I)$. By Proposition 3.21, $\mathbb{K}(I)$ is a set of knots that is founded w.r.t. P and $st(G)$. Due to the definition of \mathbb{K}_P and Proposition 3.30, we have that \mathbb{K}_P is compatible with $st(G)$. It remains to show that (ii) in (B) holds. Consider $reach_A(\mathbb{K}(I))$. Due to the fact that $A(t) \in I$ and the construction of $reach_A(\mathbb{K}(I))$, for some constant c there exists $K \in \mathbb{K}(I)$ such that $st(G, c) \approx st(K, \mathbf{x})$ and $K \in reach_A(\mathbb{K}(I))$. Due to the definition of \mathbb{K}_P , $\mathbb{K}(I) \subseteq \mathbb{K}_P$. Therefore $K \in \mathbb{K}_P$. Moreover, it trivially holds that $K \in reach_A(\mathbb{K}(I))$ implies $K \in reach_A(\mathbb{K}_P)$. Therefore, (ii) holds.

Suppose (B) holds. The facts that $G \in SM(\mathbb{gp}(P))$, and that \mathbb{K}_P is compatible with $st(G)$ imply that $\mathcal{F}(G, \mathbb{K}_P) \neq \emptyset$ and each $I \in \mathcal{F}(G, \mathbb{K}_P)$ is a stable model of P (see Theorem 3.26). The condition (ii) and the construction of forest-shaped interpretations ensure that some $I \in \mathcal{F}(G, \mathbb{K}_P)$ contains an atom $A(t)$, where t is some term. \square

PROOF OF THEOREM 5.16. Similar to the proof of Theorem 5.13, and thus omitted. \square

A.4 Open Queries

PROOF OF PROPOSITION 5.19. For the “only if” direction, suppose a ground term t is such that $P \models_c A(t)$. Suppose c is the constant of t , and $t = f_n(\dots f_1(c)\dots)$. Let t_0, \dots, t_n be the list of subterms of t ordered w.r.t. increasing term depth, i.e., $t_0 = c$ and $t_i = f_i(\dots f_1(c)\dots)$, where $i \in \{1, \dots, n\}$.

Define the sequence $s = [\frac{L_0}{c}, \frac{L_1}{f_1}, \dots, \frac{L_n}{f_n}]$, where $L_i = \{K_{\downarrow \mathbf{x}} \mid I \in SM(P), K = I \cap \mathcal{HB}_{t_i}\}$, for $i \in \{0, \dots, n\}$. We verify that s is convergent.

Since P is consistent, each L_i is a nonempty subset of \mathbb{K}_P . Since $P \models_c A(t)$, the sequence s trivially satisfies the conditions (1) and (3) in Definition 5.18. Suppose (2) is not satisfied. Then there exists some $j \in \{1, \dots, n\}$, some $K \in L_{j-1}$ and some $K' \in \mathbb{K}_P$ such that $st(K, f_j(\mathbf{x})) \approx st(K', \mathbf{x})$ and $K' \notin L_j$. Take the smallest index j for which the statement above holds. Then there exists a sequence K_0, \dots, K_{j-1}, K_j

of knots in \mathbb{K}_P such that the sequence $N = [\frac{K_0}{t_0}, \dots, \frac{K_{j-1}}{t_{j-1}}, \frac{K_j}{t_j}]$ has the following properties:

- $K_i \in L_i$ for each $i \in \{0, \dots, j-1\}$, while $K_j \notin L_j$;
- $\text{st}(K_i, f_{i+1}(\mathbf{x})) \approx \text{st}(K_{i+1}, \mathbf{x})$ for each $i \in \{0, \dots, j-1\}$.

Let $S = \text{st}(K_0, \mathbf{x})$. Due to the definition of trees, we know that there exists a tree T induced by \mathbb{K}_P with root S such that $N \in T$. Consider the stable model $I \in \text{SM}(P)$ where $\text{st}(I, c) \approx S$. Such I must exist due to the way we defined L_0 . By the semantic characterization (see Theorem 3.34), I can be represented as $I = \text{ffa}(I) \cup (T^{c_1})_{\downarrow} \cup \dots \cup (T^{c_n})_{\downarrow}$, where $\{c_1, \dots, c_n\}$ is the set of all constants of P , and each T^{c_i} is a tree induced by \mathbb{K}_P with root $\text{st}(\text{ffa}(I), c_i)$. W.l.o.g., $c_1 = c$. Simply define $I' = \text{ffa}(I) \cup (T)_{\downarrow} \cup (T^{c_2})_{\downarrow} \cup \dots \cup (T^{c_n})_{\downarrow}$. By Theorem 3.26, we have that I' is also a stable model of P . We arrive at a contradiction to the assumption that $K_j \notin L_j$. Indeed, $K_j = (I' \cap \mathcal{HB}_{t_j})_{\downarrow \mathbf{x}}$, and, due to the definition of s , $K_j \in L_j$.

For the other direction we show that the failure of (A) implies the failure of (B). Suppose for each term t , $P \not\models_c A(t)$. Furthermore, assume there exists a converging sequence $s = [\frac{L_0}{c}, \frac{L_1}{f_1}, \dots, \frac{L_n}{f_n}]$ for A , where $L_0 = \text{seeds}(c, P)$. First, we reconstruct the term encoded in the sequence. Let $t_0 = c$, while t_n is defined inductively as $t_i = f_i(t_{i-1})$, where $1 \leq i \leq n$. Consider the term t_n . By assumption, there exists a model I of P such that $A(t_n) \notin I$. There are two possibilities.

- a) $t_i \in I$, for each $i \in \{0, \dots, n\}$. Due to the definition of \mathbb{K}_P and the fact that $\mathbb{K}(I)$ is founded, we have that each $K_i = (\mathcal{HB}_{t_i} \cap I)_{\downarrow \mathbf{x}}$ is in \mathbb{K}_P , where $0 \leq i \leq n$. By assumption, we have $K_0 \in L_0$. The condition (2) in Definition 5.18 implies that $K_n \in L_n$. Since $A(\mathbf{x}) \notin K_n$, we have that s is not a converging sequence for A due to violation of (3) in Definition 5.18.
- b) For some i , where $0 < i \leq n$, we have $t_i \notin I$. Note that $t_0 \in I$ since it is a constant. Take the smallest m , where $0 \leq m < n$, such that $t_{m+1} \notin I$. As it was argued, each $K_i = (\mathcal{HB}_{t_i} \cap I)_{\downarrow \mathbf{x}}$ is in \mathbb{K}_P , where $0 \leq i \leq m$. By assumption, we have $K_0 \in L_0$. The condition (2) in Definition 5.18 implies that $K_m \in L_m$. Since $K_m = (\mathcal{HB}_{t_m} \cap I)_{\downarrow \mathbf{x}}$ and $t_{m+1} \notin I$, we have that $f_m(\mathbf{x}) \notin \text{succ}(K_m)$. We have that s is not a converging sequence for A due to violation of (1) in Definition 5.18.

In both cases s is a converging sequence for A , which contradicts the assumption. \square

DEFINITION A.1. A deterministic Turing machine (DTM) is a quadruple (S, Σ, δ, s_0) , where S is a finite set of states, Σ is a finite alphabet, δ is a transition function, and $s_0 \in S$ is the initial state. The transition function δ is a partial function

$$\delta : S \times \Sigma \rightarrow (S \cup \{\text{accept}\}) \times \Sigma \times \{-1, 0, +1\},$$

where *accept* is a new state not occurring in S . We assume a special symbol b in Σ that stands for blank symbol. By I_k we denote the k th symbol in the input string $I = I_0, \dots, I_{|I|-1}$.

PROOF OF THEOREM 5.22 (EXPSpace-HARDNESS). Consider a language L over an alphabet Σ in EXPSpace. Then there is a Turing machine $M = (S, \Sigma, \delta, s_0)$ as in Definition A.1 that decides membership of a given word I in L on a tape whose

length is bounded by an exponential in the size of I . We construct a \mathbb{FD} program $P(M, I)$ of size polynomial in M and I such that acceptance of I by M is equivalent to the existence of an answer for an open query $\lambda x.A(x)$ under cautious entailment.

For convenience, we assume here that I is not the empty word. Suppose the number of cells (the space) used by M on the input I is bounded by $m = 2^{as}$, where as is polynomial in the size of I . The reduction relies on keeping two addresses of the cells in the work tape, each of which is represented using $as = \log_2 m$ bits. The first address is the position of the read/write (r/w) head, which is encoded by the unary predicate symbols $rwpos_0^b, \dots, rwpos_{as}^b, b \in \{0, 1\}$. For each bit of the address, we dedicate two symbols and will ensure that exactly one of them holds for each term. In our encoding, terms will represent stages reached in the computation of the machine on some path. Similarly, the second address is the one of the *observed cell*, which is encoded by the unary predicate symbol $opos_0^b, \dots, opos_{as}^b, b \in \{0, 1\}$. Intuitively, the observed cell is the single cell of the machine for which the correct state transition will be ensured by the program. By non-deterministically generating all cells for observation in parallel, and exploiting the properties of cautious entailment of open queries we will ensure that accepting computations of M (represented by terms) can be singled out.

We sketch the construction of the program $P(M, I)$ in steps. We need rules for checking the equality of the r/w head address and the address of the observed cell. To this end, for a bit b , let $\bar{b} = 1 - b$ denote the complement of b . For the comparison of separate bits in the two addresses, we add the following rule

$$equ_i(x) \leftarrow opos_i^b(x), rwpos_i^b(x) \quad \text{for all } i \in \{0, \dots, as\} \text{ and } b \in \{0, 1\}. \quad (2)$$

The equality of two addresses at some point of computation is then expressed easily by the rule

$$rwoequ(x) \leftarrow equ_0(x), \dots, equ_{as}(x). \quad (3)$$

The inequality is also easily expressed by the rules

$$nonequ(x) \leftarrow opos_i^b(x), rwpos_i^{\bar{b}}(x) \quad (4)$$

for all $i \in \{0, \dots, as\}$ and $b \in \{0, 1\}$.

We move to the representation of the initial configuration of the machine, which we do from the perspective of an observed cell. To this end, we add, for $0 \leq i \leq as$, the facts

$$rwpos_i^0(st) \leftarrow, \quad (5)$$

$$state_{s_0}(st) \leftarrow, \quad (6)$$

$$opos_i^1(st) \vee opos_i^0(st) \leftarrow. \quad (7)$$

Intuitively, (5) sets the position of the r/w head to the left most cell and (6) set the machine into the start state, while (7) non-deterministically chooses an observed cell of the tape. To represent the content of each observed cells in the initial configuration, we proceed as follows.

For each symbol $\alpha \in \Sigma$, we use a designated unary predicate symbol $symbol_\alpha$. Let $n \geq 0$ be the position of the last symbol of I written on the tape, i.e., $I = I_0 I_1 \dots I_n$ is on positions $0, \dots, n$. For each position $i \leq n$ with binary representation

$i = b_0 \cdots b_{as} = i$ and $\alpha = I_i$, we add the rule

$$symbol_\alpha(st) \leftarrow opos_0^{b_0}(st), \dots, opos_{as}^{b_{as}}(st). \quad (8)$$

For all other positions, the symbols are blank. Assuming that $n = b_0^* \cdots b_{as}^*$ in binary, we express this with rules

$$symbol_b(st) \leftarrow opos_1^{b_1^*}(x), \dots, opos_{j-1}^{b_{j-1}^*}(x), opos_j^1(x), \quad (9)$$

for all $j \in \{0 \dots, as\}$ such that $b_j^* = 0$.

This describes the initial configuration; note that it is captured by the whole set of models for the program described so far. Although each model captures only the content of one (the observed) cell, the contents of the whole work tape is entirely captured as the addresses of the observed cells cover the whole work space of M .

To encode the transitions, it is handy to view δ as a table. For each tuple $t = \langle s, \alpha, s', \alpha', D \rangle$ such that $\delta(s, \alpha) = \langle s', \alpha', D \rangle$, we use a function symbol \bar{t} and define the following rules:

$$next(x, \bar{t}(x)) \leftarrow rwoequ(x), state_s(x), symbol_\alpha(x), \quad (10)$$

$$next(x, \bar{t}(x)) \leftarrow nonequ(x), state_s(x), \quad (11)$$

$$state_{s'}(\bar{t}(x)) \leftarrow next(x, \bar{t}(x)), \quad (12)$$

$$symbol_{\alpha'}(\bar{t}(x)) \leftarrow rwoequ(x), next(x, \bar{t}(x)), \quad (13)$$

$$move_D(\bar{t}(x)) \leftarrow next(x, \bar{t}(x)). \quad (14)$$

The rules above are explained as follows. If the r/w head is at the position of the observed cell, and the symbol and the state are correct for the transition, the transition is made (10). If the r/w head is not at the position of the observed cell, the transition is made blindly (11). The single case where the transition is not made is if the r/w head is at the position of the observed cell, but either the symbol or the state is not the right one. The rule (12) sets the new state, while (13) sets the new symbol of the observed cell. The rule (14) triggers the movement of the r/w head. The effect of $move_D$ is explained next. Moving the r/w head boils down to adding or subtracting one bit from the address. To this end, we use unary predicates $shift_0^b, \dots, shift_{as}^b$, $b \in \{0, 1\}$, to simulate the values of the carry bit. When the r/w head position changes, the last bit should be inverted. This is stated by the rules

$$shift_{as}^1(x) \leftarrow move_{+1}(x), \quad (15)$$

$$shift_{as}^1(x) \leftarrow move_{-1}(x), \quad (16)$$

$$shift_{as}^0(x) \leftarrow move_0(x). \quad (17)$$

The position of r/w head after shifting is then defined by the following rules for

each $j \in \{0, \dots, as\}$, $j' \in \{1, \dots, as\}$, and $b \in \{0, 1\}$:

$$rwpos_j^{\bar{b}}(y) \leftarrow shift_j^1(y), rwpos_j^b(x), next(x, y), \quad (18)$$

$$rwpos_j^b(y) \leftarrow shift_j^0(y), rwpos_j^b(x), next(x, y), \quad (19)$$

$$shift_{j'-1}^b(y) \leftarrow move_{+1}(y), shift_{j'}^1(y), rwpos_{j'}^b(x), next(x, y), \quad (20)$$

$$shift_{j'-1}^{\bar{b}}(y) \leftarrow move_{-1}(y), shift_{j'}^1(y), rwpos_{j'}^b(x), next(x, y), \quad (21)$$

$$shift_j^0(x) \leftarrow move_0(x). \quad (22)$$

Furthermore, we have to state that the address of the observed cell does not change, i.e., is fixed for a model. This is expressed by the rules

$$opos_i^b(y) \leftarrow opos_i^b(x), next(x, y), \quad (23)$$

for each $i \in \{0, \dots, as\}$ and $b \in \{0, 1\}$. Finally, we ensure that the symbol written in the observed cell does not change if it is not affected by the transition. This is expressed by the following inertia rule for each $\alpha \in \Sigma$:

$$symbol_\alpha(y) \leftarrow nonequ(x), symbol_\alpha(x), next(x, y). \quad (24)$$

This completes the description of the program $P(M, I)$. It is not hard to see that $P(M, I)$ has exactly $m = 2^{as}$ minimal models (and thus stable models, as in $P(M, I)$ no negation occurs) that are induced by different choices of the position of the observed cell. Let R^0, \dots, R^{m-1} be these models ordered with respect to the position of the observed cell, i.e., R^0 is the one for first position 0 while R^{m-1} is the one for the last position $m - 1$.

Without loss of generality, we view a run of M on an input I as a sequence t_1, \dots, t_n of transitions, and assume that it is always non-empty. The run is accepting, if after performing t_n , the machine enters the accepting state *accept*. We establish the following lemmas.

LEMMA A.2. *If the machine M accepts the input I on the run t_1, \dots, t_n , $n \geq 1$, then $P(M, I) \models_c state_{accept}(u)$, where $u = \bar{t}_n(\dots \bar{t}_1(st) \dots)$.*

PROOF. Suppose that $I^0 = Ib \dots b$ is the word describing the initial tape contents, and that after executing the transitions t_1, \dots, t_i , (i) I^i is the word given by the tape contents, (ii) s^i is the state of the machine, and (iii) pos^i is the position of the r/w head.

We show that for each R^w , $w \in \{0, \dots, m - 1\}$, we have $state_{accept}(u) \in R^w$. To this end, we show that in R^w the content of the observed cell w , the state, and the r/w head position are correctly reflected through the computation. More formally, let $u_0 = st$, and $u_i = \bar{t}_i(u_{i-1})$, where $0 < i \leq n$. Then we argue that, for each $j \in \{0, \dots, n\}$, (i) $symbol_\alpha(u_j) \in R^w$ whenever $\alpha = I_w^j$, i.e., α is written in cell w , (ii) $state_{s_j}(u_j) \in R^w$, and (iii) pos^j is, encoded, in binary, by the atoms $rwpos_i^b(u_j) \in R^w$, $0 \leq i \leq as$. Note that this will prove the lemma, since $s^n = accept$.

We proceed by induction on $j \geq 0$. The base case $j = 0$ is clear by the encoding of the initial word (rules (8) and (9)), the initial r/w head position (facts (5)) and the initial state (fact (6)).

For the inductive case, assume the claim holds for $0 \leq j < n$ and consider $j + 1$. By the induction hypothesis, $symbol_\alpha(u_j) \in R^w$, $state_{s_j}(u_j) \in R^w$, and pos^j is described by the atoms $rwpos_i^b(u_j) \in R^w$. There are now, by the rules (2) – (4) two disjoint cases: either $nonequ(u_j) \in R^w$ or $rwoequ(u_j) \in R^w$. In the former case, $next(u_j, t_{j+1}(u_j)) \in R^w$ by the rule (11); by the rule (24), we then have $symbol_\alpha(u_{j+1}) \in R^w$. In the latter case, $next(u_j, t_{j+1}(u_j)) \in R^w$ by the rule (10); by the rule (13), we then have $symbol_{\alpha'}(u_{j+1}) \in R^w$. In both cases, R^w contains $symbol_\alpha(u_{j+1})$ where $I_w^{i+1} = \alpha$. Hence (i) holds for $j + 1$.

As for (ii), as we have $next(u_j, t_{j+1}(u_j)) \in R^w$, by the rule (12) we have $state_{s_{i+1}}(u_{i+1}) \in R^w$, and thus (ii) holds for $j + 1$. Finally, the rules (14) and (15) – (22) effect that atoms $pos_i^b(u_{j+1})$ which correctly represent pos^{j+1} are derived. Hence, (iii) holds for $j + 1$. \square

LEMMA A.3. *If $P(M, I) \models_c \lambda x.state_{accept}(x)$, then there exists an accepting run of M .*

PROOF. Suppose $P(M, I) \models_c state_{accept}(u)$. By assumption, the initial state is not *accept* and thus $u = \bar{t}_n(\dots \bar{t}_1(st)\dots)$, where $n \geq 1$. Let $u_0 = st$, and $u_i = \bar{t}_i(u_{i-1})$, where $0 < i \leq n$. Then, in each model R^w , we must clearly must have $next(u_{i-1}, t_i(u_{i-1}))$ for each $0 < i \leq n$ (otherwise, $state_{accept}(u_n)$ would not be contained in R^w).

For each $i \in \{0, \dots, n\}$, define (i) the word $I^i = \alpha_0 \cdots \alpha_{m-1}$ where α_j is such that $symbol_{\alpha_j}(u_i) \in R^j$, $0 \leq j < m$, (ii) s^i as the state s such that $state_s(u_i) \in R^w$, and (iii) pos^i as the integer which, in binary, is encoded by the facts $rwpos_i^{b_i}(u_j) \in R^w$, i.e., $pos^i = b_0 \cdots b_{as}$, where $w \in \{0, \dots, m-1\}$ is arbitrary.

We claim that each I^i , s^i , and pos^i is well-defined and is the tape contents, state, and r/w head position, respectively, after the partial run t_1, \dots, t_i of M on the input I . Since $s^n = accept$, this will prove the lemma.

The proof is by induction on $i \geq 0$. For the base case $i = 0$, by construction, I^0 clearly is the initial tape contents, $s^0 = s_0$, and $pos^0 = 0$ by the facts and rules (5) – (9). Suppose the claim holds for $0 \leq i < n$ and consider $i + 1$. Assume $t_{i+1} = \langle s, \alpha, s'\alpha', D \rangle$. Since we have $next(u_i, t_{i+1}(u_i))$ in each R^w , we must have $state_{s'}(u_{i+1})$ in R^w by rule (12); since no other fact $state_{s''}(u_{i+1})$ can be in R^w , s^{i+1} is well-defined. Furthermore, we must have $state_s(u_i)$ in R^w and either (a) $rwoequ(u_i) \in R^w$ or (b) $nonequ(u_i) \in R^w$; by the induction hypothesis and the rules (2) – (4), (a) is the case if $pos^i = w$ and (b) if $pos^i \neq w$. In case (a), we must have $symbol_\alpha(u_i) \in R^w$ and $symbol_{\alpha'}(u_{i+1}) \in R^w$ by rule (13), and in case (b) $symbol_\alpha(u_{i+1}) \in R^w$ by rule (24), where $symbol_\alpha(u_i) \in R^w$. Since no other facts $symbol_{\alpha''}(u_{i+1})$ can be in R^w , I^{i+1} is well-defined. Finally, we must have $move_D(u_{i+1})$ in R^w by rule (14); by the induction hypothesis and the rules (15) – (22), we have facts $rwpos_j^{b_j}(u_{i+1})$ in R^w , $0 \leq j \leq as$, such that b_0, \dots, b_{as} represents $pos^i + D = pos^{i+1}$ in binary.

Summing up, I^{i+1} , s^{i+1} , and pos^{i+1} are all well-defined and encode tape contents, state, and r/w head position, respectively, after the partial run t_1, \dots, t_{i+1} of M on the input I , which concludes the induction step. \square

As $P(M, I)$ and $\lambda x.state_{accept}(x)$ are constructible in polynomial time from M and I , from Lemmas A.2 and A.3 the claimed EXPSPACE-hardness result follows

for FD , FDN , and FDNC ; by replacing the disjunctive guessing rules (7) with unstratified rules $\text{opos}_i^1(st) \leftarrow \text{not opos}_i^0(st)$; $\text{opos}_i^0(st) \leftarrow \text{not opos}_i^1(st)$, we obtain the result for FN and FNC .

A.5 Reasoning in FN

PROOF OF PROPOSITION 6.2. Let I be a stable model of $\text{fr}(P)$. The properties (1), (2) and (3) hold by the construction of $\text{fr}(P)$, i.e., due to the fact that I is a stable model that satisfies (F1)-(F4) rules of $\text{fr}(P)$. Suppose I does not satisfy (4), i.e., for some term $t \in I$ and some unary predicate A of P , either (a) $\{A(t), \bar{A}(t)\} \cap I = \emptyset$, or (b) $\{A(t), \bar{A}(t)\} \subseteq I$. In case (a), we have $\{A(t) \leftarrow \text{Dom}(t); \bar{A}(t) \leftarrow \text{Dom}(t)\} \subseteq \text{fr}(P)^I$. Since $\text{Dom}(t) \in I$, I is not a model of $\text{fr}(P)^I$. In case (b), by construction of $\text{fr}(P)$, there is no rule in $\text{fr}(P)^I$ that has head $A(t)$ or $\bar{A}(t)$, and hence I is not a minimal model of $\text{fr}(P)^I$. In both cases, we arrive to a contradiction to the assumption that I is a stable model of $\text{fr}(P)$. Analogously to the argument for (4), one can show that the property (5) holds.

For the other direction, assume an interpretation I of $\text{fr}(P)$ for which the given properties hold. It is easy to see that such an interpretation satisfies each of the rules in $\text{fr}(P)^I$. We verify that I is a minimal model of $\text{fr}(P)^I$. By the construction of $\text{fr}(P)$, each minimal model of $\text{fr}(P)$ has to satisfy (1), (2) and (3). Therefore, if I is not a minimal model of $\text{fr}(P)^I$, there should exist a model $H \subset I$ of $\text{fr}(P)^I$ for which (4) or (5) does not hold. We arrive to a contradiction. Due to the rules of types (F5-F8) in $\text{fr}(P)$, H cannot be a model of $\text{fr}(P)^I$. \square

PROOF OF PROPOSITION 6.4. Suppose P is consistent, and I is a minimal model of P . We know that I is forest-shaped. Let J be a Herbrand interpretation for $\text{tf}(P)$ defined as the smallest set of atoms satisfying the following conditions:

- a) $I \subseteq J$,
- b) $S(c, d) \in J$, for each pair c, d of constants of P ,
- c) if $t \in J$, then $\text{Dom}(t) \in J$,
- d) if $t \in J$ and f is a function symbol of P , then $S(t, f(t)) \in J$,
- e) if $t \in J$ and $A(t) \notin J$, then $\bar{A}(t) \in J$, and
- f) if $S(s, t) \in I$ and $R(s, t) \notin I$, then $\bar{R}(s, t) \in J$,

where A and R are predicate symbols of P . We show that J is a stable model of $\text{tf}(P)$. Assume that it is not the case. There are two possibilities.

- 1) J is not a model of $\text{tf}(P)^J$. Since $\text{fr}(P) \subseteq \text{tf}(P)$, we have $\text{fr}(P)^J \subseteq \text{tf}(P)^J$. From the construction of J it follows that J is a model of $\text{fr}(P)^J$. Then $\text{tf}(P)^J$ must contain some ground rule

$$C(\vec{v}_1) \leftarrow Q_1(\vec{v}_1), \dots, Q_m(\vec{v}_m), \bar{W}_1(\vec{t}_1), \dots, \bar{W}_n(\vec{t}_n)$$

such that $(\star) \{Q_1(\vec{v}_1), \dots, Q_m(\vec{v}_m), \bar{W}_1(\vec{t}_1), \dots, \bar{W}_n(\vec{t}_n)\} \subseteq J$ and $C(\vec{v}_1) \notin J$. By construction, P contains the rule $W_1(\vec{t}_1) \vee \dots \vee W_n(\vec{t}_n) \leftarrow Q_1(\vec{v}_1), \dots, Q_m(\vec{v}_m)$, where $n, m \geq 0$. Since I is a model of P , then either (a) $\{Q_1(\vec{v}_1), \dots, Q_m(\vec{v}_m)\} \not\subseteq I$ or (b) $\{W_1(\vec{t}_1), \dots, W_n(\vec{t}_n)\} \cap I \neq \emptyset$. In case (a), by the definition of J , (\star) does not hold. In case (b), for some $W_i(\vec{t}_i)$ of the rule, $\bar{W}_i(\vec{t}_i) \notin J$ and, hence, (\star) does not hold.

Generating time:
$Time(st) \leftarrow$
$N(x, f(x)) \leftarrow Time(x)$
$Time(y) \leftarrow N(x, y)$
Initial configuration:
$Sym_{\alpha, \pi}(st) \leftarrow$ for $0 \leq \pi < I $ such that $\alpha = I_\pi$
$Sym_{b, \pi}(st) \leftarrow$ for $ I \leq \pi \leq sb(I)$
$Cur_0(st) \leftarrow$
$St_{s_0}(st) \leftarrow$
Transition $\delta(s, \sigma) = \langle s', \sigma', d \rangle$, where $0 \leq \pi \leq sb(I)$
$Sym_{\sigma', \pi}(y) \leftarrow N(x, y), St_s(x), Sym_{\sigma, \pi}(x), Cur_\pi(x)$
$St_{s'}(y) \leftarrow N(x, y), St_s(x), Sym_{\sigma, \pi}(x), Cur_\pi(x)$
$Cur_{\pi+d}(y) \leftarrow N(x, y), St_s(x), Sym_{\sigma, \pi}(x), Cur_\pi(x)$
Inertia rules, where $0 \leq \pi < \pi' \leq sb(I)$:
$Sym_{\sigma, \pi}(y) \leftarrow N(x, y), Sym_{\sigma, \pi}(x), Cur_{\pi'}(x)$
$Sym_{\sigma, \pi'}(y) \leftarrow N(x, y), Sym_{\sigma, \pi'}(x), Cur_\pi(x)$

Table IV. $\mathbb{F}\mathbb{C}$ program $P(T, I)$ for simulating a DTM T on input I .

- 2) J is a model but is not a minimal model of $\text{tf}(P)^J$. Since $\text{fr}(P) \subseteq \text{tf}(P)$, we have $\text{fr}(P)^J \subseteq \text{tf}(P)^J$. Then we also have that J is a model of $\text{fr}(P)^J$, but is not minimal. However, by construction of J , we have $J \subseteq \mathcal{HB}^{\text{fr}(P)}$ and J satisfies the conditions in Proposition 6.2, and hence by the same proposition, J must be a minimal model of $\text{fr}(P)^J$. Contradiction.

For the other direction, let $I \in SM(\text{tf}(P))$. Let J be the restriction of I to the predicates of P . Suppose $J \not\models P$. Then $\text{Ground}(P)$ contains a rule

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

where $n, m \geq 0$, such that $\{Q_1(\vec{v}_1), \dots, Q_m(\vec{v}_m)\} \subseteq J$ and $\{W_1(\vec{t}_1), \dots, W_n(\vec{t}_n)\} \cap J = \emptyset$. By construction, $\text{Ground}(\text{tf}(P))$ contains

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

By hypothesis, $I \in SM(\text{tf}(P))$. Clearly, $C(t_1) \notin I$ (otherwise, $I \notin MM(\text{tf}(P)^I)$ as no rule in $\text{tf}(P)^I$ would have $C(t_1)$ in the head). Hence, $\text{tf}(P)^I$ contains the rule resulting from r by removing $\text{not } C(\vec{t}_1)$. Since $\{Q_1(\vec{v}_1), \dots, Q_m(\vec{v}_m)\} \subseteq I$ and $I \models \text{tf}(P)^I$, it follows that $\bar{W}_i(\vec{t}_i) \notin I$ for some $i \in \{1, \dots, n\}$. As I is forest-shaped, by the rules in $\text{fr}(P)$ we have $W_i(\vec{t}_i) \in I$, and thus $W_i(\vec{t}_i) \in J$. However, this contradicts that $\{W_1(\vec{t}_1), \dots, W_n(\vec{t}_n)\} \cap J = \emptyset$. \square

A.6 Reasoning in $\mathbb{F}\mathbb{C}$

LEMMA A.4. *Deciding whether a given $\mathbb{F}\mathbb{C}$ program is consistent is PSPACE-hard.*

PROOF. Let \mathcal{L} be a language in PSPACE, and let T be a DTM which decides whether a given word I is in \mathcal{L} within space $sb(I)$ that is polynomial in $|I|$. The computation of T on I can be simulated by an \mathbb{F} program $P(T, I)$ (see Table A.6). Due to construction, we can use a single constraint to decide whether $I \in \mathcal{L}$. It is easy to see that $I \in \mathcal{L}$ iff $P(T, I) \cup \{\leftarrow St_{\text{accept}}(x)\}$ is inconsistent.

As the translation is clearly polynomial in the size of T and I , deciding $I \in \mathcal{L}$ is reducible in polynomial time to consistency checking of an $\mathbb{F}\mathbb{C}$ program. \square

A.7 Reasoning in \mathbb{F} and \mathbb{FD}

PROOF OF THEOREM 6.16. If (i) holds, then, due to Theorem 3.13, we can easily define G and L such that the conditions in (ii) are satisfied. On the other hand, if (ii) is satisfied, the fact that for each $G \in MM(\mathbf{gp}(P))$ there is some $M \in MM(P)$ such that $G = ffa(M)$ and Theorem 3.13 imply that a minimal model of P such that $A(t) \in P$ for some term t is constructible. Indeed, take some M for G as described. By Theorem 3.13, M is forest-shaped. As L is founded w.r.t. P and $\{\mathbf{st}(G, c)\}$, the tree in M rooted at c can be replaced by some tree that is built with instances of knots from L only, and such that some instance of a knot K containing $A(\mathbf{x})$ is used. As the resulting model I is forest-shaped, by Theorem 3.13 we obtain that $I \in SM(P)$. \square

LEMMA A.5. For \mathbb{F} programs, brave entailment of unary existential queries is PSPACE-hard.

PROOF. Recall the program $P(T, I)$ in Table A.6 that simulates a computation of a DTM T on input I . Note that it is an \mathbb{F} program and has size polynomial in the size of T and I . To check whether T accepts I , we can pose the brave query whether $St_{accept}(t)$ is in the minimal model of $P(T, I)$ for some term t , i.e., T accepts I iff $P(T, I) \models_b \exists x. St_{accept}(x)$. \square

PROOF OF PROPOSITION 6.19. For the only-if direction of the first claim, assume we have a stable model I of P such that $A(t) \in I$. Due to Theorem 3.13, we can simply define $G = ffa(I)$ and $K_i = \mathcal{HB}_{s_i} \cap I$, where $1 \leq i \leq n$. For the if direction, since for \mathbb{FD} programs each $G \in SM(\mathbf{gp}(P))$ is extendible to some $M \in SM(P)$, we can similarly as in the proof of Theorem 6.16 construct using Theorem 3.13 a stable model of P containing $A(s_n)$; simply start with $G \cup K_1 \cup \dots \cup K_n$ and extend the set with the necessary stable knots.

For the only-if direction of the second claim, the argument is as for the first claim. If $I \in SM(P)$ such that $A(t) \notin I$, then, by Theorem 3.13, we can easily define G and the sequence of knots. Again, take $G = ffa(I)$ and $K_i = \mathcal{HB}_{s_i} \cap I$, where $1 \leq i \leq n$. For the if direction, let I be the unique stable model of P . Let $K'_i = \mathcal{HB}_{s_i} \cap I$, where $1 \leq i \leq n$. Due to Theorem 3.13, we have $I^c \subseteq G$ and $K'_i \subseteq K_i$, where $1 \leq i \leq n$. Hence, $A(s_n) \notin I$. \square

B. APPLICATION: REASONING ABOUT ACTIONS AND PLANNING

In Section 5, we have already encountered an application of \mathbb{FDNC} programs to Description Logics. In this section, we consider a further application of \mathbb{FDNC} programs in the area of reasoning about actions and planning; recall that non-monotonic logic programs under answer set semantics have been widely used in this area. In particular, we apply \mathbb{FDNC} programs to planning under incomplete knowledge and non-deterministic action effects, based on the expressive action language \mathcal{K} [Eiter et al. 2004].

Transition-based action formalisms are based on languages for describing legal transitions between states of the world which happen due to the execution of actions by some agent. A classical problem is that of *plan existence*, which consists of finding a sequence of actions that leads the agent from an initial to some desired goal state of the world. Apart from this, many problems have been considered,

including *plan verification* (i.e., whether a given candidate plan is good to reach a goal state) and *temporal projection* (i.e., reasoning about the hypothetical future if a sequence of action would be taken); as for the concerns of this paper, we refer to [Baral 2002] for background and a study of these problems based on logic programs under answer set semantics.

As for temporal projection in Example 3.2, view *grow*, *cell₁*, *cell₂*, and *die* as actions and *Young*, *Warm*, *Cold*, and *Mature* as fluents. As seen, if the sequence of actions *grow* and *cell₁* would happen, the fluent *Young* would be possibly true, as $Young(cell_1(grow(b)))$ is bravely entailed by the program. On the other hand, *Young* is not necessarily true after this action sequence. Indeed, using similarly as in Proposition 6.13 an auxiliary fact $C_0(b)$. and rules $R(x, grow(x)) \leftarrow C_0(x)$; $C_1(y) \leftarrow C_0(x), R(x, y)$; $R(x, cell_1(x)) \leftarrow C_1(x)$; and $\leftarrow R(x, y), not\ Change(x, y)$, we can eliminate those stable models of P^{ex} which do not correspond to the occurrence of this sequence; the resulting program P' does not cautiously entail $Young(cell_1(grow(b)))$, as it has a stable model which does not contain this atom. In this scenario, planning seems not to make sense (as bacteria can't really take actions), and we thus consider a different one.

For modeling planning domains, several dedicated action languages have been proposed that are rooted in knowledge representation formalisms, including \mathcal{A} [Gelfond and Lifschitz 1992] (which was extensively studied in [Baral 2002]), \mathcal{C} [Giunchiglia and Lifschitz 1998], and \mathcal{K} [Eiter et al. 2004]. The latter, which we consider in the sequel, is based on the principles of logic programming under the stable model semantics. In contrast to the other languages, \mathcal{K} allows to describe transitions between knowledge states, which are incompletely described states of the world. The availability of non-monotonic negation in \mathcal{K} makes the formalism suitable for common-sense and heuristic reasoning in planning applications.

In \mathcal{K} , a *planning domain PD* is a set of rules that describes the initial state I and legal transitions.⁶ At the core, it distinguishes two kinds of predicates: *fluents* and *actions*. A *state* is given by a set of ground fluent literals which are known to hold at a particular stage. A *goal G* is a set of ground fluent literals, each of which can also be default negated.

An *optimistic (or credulous) plan* for a given planning domain PD and a goal G is a sequence of action occurrences $\langle \mathcal{A}_1, \dots, \mathcal{A}_n \rangle$, $n \geq 0$, that legally transforms the initial state I into some state that satisfies the goal G , i.e., for some sequence of states, we have (i) $S_0 = I$, (ii) each $S_i, \mathcal{A}_{i+1}, S_{i+1}$ is a legal transition, and (iii) S_n satisfies G . For our concerns, \mathcal{A}_i is a single action.

In case of non-deterministic action effects or incomplete information about the initial state, executing an optimistic plan does not necessarily establish the goal. This is ensured by *secure plans* (also known as *conformant plans*), which are optimistic plans such that, regardless of such incompleteness and non-deterministic

⁶We consider here merely a simplified version of \mathcal{K} that contains the salient elements; missing features like static predicates, typing and others can be added easily on top. Furthermore, we assume that actions are not executed in parallel (parallel execution may be encoded using designated action symbols), that at each stage some action has to be taken to move on (thus passage of time would have to be modeled explicitly by an action), and that taking an executable action always results in a follow up state. Technically, such planning domains are *proper* and more general than *plain* ones in the sense of [Eiter et al. 2004].

action effects, all actions can be executed and the goal is established after the last action.

The legal state transitions are defined in \mathcal{K} in terms of stable model semantics. Roughly speaking, this is accomplished using a set of statements, similar to logic program rules, which describe the value of the fluents in the successor state S' depending on the previous state S , the action A that was taken, and the value of other fluents in S' . Because of this similarity, planning problems in \mathcal{K} can be naturally encoded into $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ programs. Via such encodings, optimistic and secure plan existence can be characterized in terms of brave entailment of existential queries and cautious entailment of open queries, respectively.

More in detail, we consider here the propositional fragment of \mathcal{K} , i.e., predicates are nullary (predicates of higher arity can be supported using higher-arity $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ from Section 7). A planning domain PD in \mathcal{K} consists of *causation rules*, *executability conditions*, and *initial state constraints*. The causation rules of propositional \mathcal{K} are of the form

$$\begin{aligned} \text{caused } D \text{ if } & B_1, \dots, B_n, \text{not } B_{n+1}, \dots, \text{not } B_k \\ \text{after } & C_1, \dots, C_m, \text{not } C_{m+1}, \dots, \text{not } C_l \\ & A_1, \dots, A_v, \text{not } A_{v+1}, \dots, \text{not } A_w \end{aligned} \quad (25)$$

$k, l, w \geq 0$, where D and $B_1, \dots, B_k, C_1, \dots, C_l$ are fluent literals, and A_1, \dots, A_w are action atoms. Intuitively, the rule (25) describes the (incomplete) knowledge state after action execution, where the knowledge depends on fluents that hold (C_1, \dots, C_m) and do not hold (C_{m+1}, \dots, C_l) in the old state, fluents that hold (B_1, \dots, B_n) and do not hold (B_{n+1}, \dots, B_k) in the new state, and actions that were executed (A_1, \dots, A_v) respectively were not executed (A_{v+1}, \dots, A_w) in parallel.⁷

The *executability conditions* in \mathcal{K} are of the form

$$\begin{aligned} \text{executable } A \text{ if } & B_1, \dots, B_n, \text{not } B_{n+1}, \dots, \text{not } B_k, \\ & A_1, \dots, A_m, \text{not } A_{m+1}, \dots, \text{not } A_l, \end{aligned} \quad (26)$$

where A, A_1, \dots, A_l are action atoms, and B_1, \dots, B_k are fluent literals, $k, l \geq 0$. Intuitively, they are the rules constraining the states for which a given action can be executed.

The initial state constraints in \mathcal{K} are of the form

$$\text{initially caused } D \text{ if } B_1, \dots, B_n, \text{not } B_{n+1}, \dots, \text{not } B_k \quad (27)$$

where D, B_1, \dots, B_k are fluent literals, $k \geq 0$. These rules describe the initial knowledge. Unconditional initial knowledge is described by the rules with an empty **if** part.

We next sketch the elements of a possible encoding of the planning domain PD into an $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ program. As in Section 7, we enhance $\mathbb{F}\mathbb{D}\mathbb{N}\mathbb{C}$ programs with “strong” negation $\neg P(\vec{x})$ [Gelfond and Lifschitz 1991], which is expressed in the core language as usual.

—For each propositional fluent symbol D , we use a unary predicate symbol D in the encoding. The meaning of $D(x)$ is that D holds at stage x . For each propositional

⁷For our concerns, we may restrict to $v \leq 1$.

action A , we use a binary predicate symbol A in the encoding. Intuitively, $A(x, y)$ means that A is executed in stage x with the resulting stage y .

- We use a unary predicate symbol S , with $S(x)$ meaning that x is a stage (or a situation). For the encoding we add the fact $S(\text{init}) \leftarrow$ denoting that the constant init is the initial stage. We also use a designated binary predicate symbol Tr to denote the transition to the next stage. For this reason, we also add $S(y) \leftarrow Tr(x, y)$.
- We adopt a function symbol f_A for each action A of the planning domain. Additionally, for each action A , we add the rule $A(x, f_A(x)) \leftarrow Exec_A(x)$ and the rule $Tr(x, y) \leftarrow A(x, y)$, where $Exec_A$ is a designated predicate name. Intuitively, the first rule “implements” the action execution, i.e., if $Exec_A$ holds at some stage x , then A is executed, which results in the follow up stage $f_A(x)$. The second rule makes Tr capture all executed transitions.

We can now state the encoding of the three types of rules of the planning domain PD .

- The causation rule (25) is transformed into the following rule:

$$\begin{aligned} D(y) \leftarrow & B_1(y), \dots, B_n(y), \text{not } B_{n+1}(y), \dots, \text{not } B_k(y), \\ & C_1(x), \dots, C_m(x), \text{not } C_{m+1}(x), \dots, \text{not } C_l(x), \\ & A_1(x, y), \dots, A_v(x, y), \text{not } A_{v+1}(x, y), \dots, \text{not } A_w(x, y), Tr(x, y) \end{aligned}$$

- The executability condition (26) is transformed into the following rule:

$$\begin{aligned} Exec_A(y) \leftarrow & S(y), B_1(y), \dots, B_n(y), \text{not } B_{n+1}(y), \dots, \text{not } B_k(y), \\ & A_1(x, y), \dots, A_v(x, y), \text{not } A_{v+1}(x, y), \dots, \text{not } A_w(x, y). \end{aligned}$$

Here, we assume for simplicity as in [Eiter et al. 2003] that there are no positive cyclic interdependencies between actions.

- The initial state constraint (27) is transformed into the following rule:

$$D(\text{init}) \leftarrow B_1(\text{init}), \dots, B_n(\text{init}), \text{not } B_{n+1}(\text{init}), \dots, \text{not } B_k(\text{init}),$$

The translation above allows to reformulate planning problems in PD as reasoning tasks for FDNC programs. A goal G in PD is an expression of the form

$$G_1, \dots, G_n, \text{not } G_{n+1}, \dots, \text{not } G_k \quad (28)$$

where each G_i is a fluent literal. For this, we add to the translation the following rule:

$$Plan(x) \leftarrow G_1(x), \dots, G_n(x), \text{not } G_{n+1}(x), \dots, \text{not } G_k(x) \quad (29)$$

where $Plan$ is a new predicate symbol. Let $P(PD, G)$ denote the resulting program.

To know whether an optimistic plan for G in PD exists, we can pose the brave query $\exists x. Plan(x)$ to the program $P(PD, G)$. Similarly, the cautious open query $\lambda x. Plan(x)$ can be posed for a secure plan. Due to the stable model semantics of both languages, it is not hard (yet technical) to show that a stable model of $P(PD, G)$ encodes a set of possible trajectories $S_0, \mathcal{A}_1, S_1, \mathcal{A}_2, \dots$ in PD , i.e., alternating sequences of states S_i and action occurrences \mathcal{A}_{i+1} such that each $S_i, \mathcal{A}_{i+1}, S_{i+1}$ is a legal transition, $0 \leq i < n$, where S_0 is any initial knowledge state; the whole set $SM(P(PD, G))$ captures all the trajectories for PD .

Further, each term t such that $P(PD, G) \models_b Plan(t)$ naturally encodes an optimistic plan for the problem, and each term t that is an answer for $\lambda x.Plan(x)$ under cautious entailment encodes a secure plan. Thus, plan correctness and security verification problems can be readily solved by the standard inference tasks $P(PD, G) \models_b Plan(t)$ and $P(PD, G) \models_c Plan(t)$.

We note at this point that deciding the existence of some secure plan (of arbitrary length) to establish a given goal G in a given \mathcal{K} action domain that conforms to the setting considered here is EXPSPACE-complete (this is well-known for a generic related action formalism [Haslum and Jonsson 1999]; the hardness part can be shown by slightly adapting the NEXPTIME-hardness proof for the problem when a prescribed plan length is part of the input [Eiter et al. 2004]).

Finally, also temporal projection with respect to an action sequence $\vec{A} = A_1, A_2, \dots, A_k$, $k \geq 1$, can be easily expressed: whether a fluent D is possibly true after hypothetically taking \vec{A} is expressed by the entailment $P(PD) \models_b d(t)$ where $t = f_{A_k}(f_{A_{k-1}} \dots (f_{A_1}(init)))$ where $P(PD)$ is $P(PD, G)$ except the rules (28) and (29). Whether D is necessarily true when \vec{A} would have happened can be expressed, using again a similar technique as in Proposition 6.13, as cautious entailment of $D(t)$ from $P(D)$ augmented with the auxiliary fact $C_0(init) \leftarrow$ and rules

$$\begin{aligned} R(x, f_{A_{i+1}}(x)) &\leftarrow C_i(x), \text{ for } 0 \leq i < k, \\ C_{i+1}(y) &\leftarrow C_i(x), R(x, y), \text{ for } 0 \leq i < k - 1, \\ &\leftarrow R(x, y), \text{ not } Tr(x, y), \end{aligned}$$

where all C_i and R are fresh predicates (this singles out the models in which \vec{a} would be taken).

Further tasks like reasoning about the initial state or observation assimilation [Baral 2002] can be similarly expressed.

EXAMPLE B.1. *Table V presents an example encoding of a propositional planning domain in \mathcal{K} into an FDN program P , which is an adaptation of the classical Yale-Shooting example [Hanks and McDermott 1987]. Here we assume three fluents *See*, *Loaded*, *Hit*, and two actions *load* and *shoot*. In the initial situation, a hunter sees a target, but his gun is not loaded (row (1)). The fluents *See* and *Loaded* are inertial, i.e., their truth values do not change unless proved otherwise (rows (2) and (3)). The hunter can load the gun only if it is unloaded, and can shoot only if the gun is loaded (rows (4) and (5)). The gun becomes loaded after loading occurs (row(6)). Finally, the hunter hits the target, if he shoots while seeing the target (row (7)). The goal in the planning domain is *Hit*, and hence the rule $Plan(x) \leftarrow Hit(x)$ is added to the encoding.*

*It is easy to see that $P \models_b \exists x.Plan(x)$, i.e., there exists a plan where the hunter hits the target and is witnessed by the term $t = \text{shoot}(\text{load}(\text{init}))$. The inertia of *See* is crucial; dropping the statement in row (3) wouldn't let us assume that the hunter still sees the target after loading the gun.*

*The term t also encodes a secure plan for the domain, i.e., t witnesses the open query $\lambda x.Plan(x)$. This becomes false when instead of sure knowledge that the gun is not loaded in the initial state, the status of the gun in the initial stage can vary freely. This situation is modeled by the two rules **caused** *Loaded* **if not** \neg *Loaded* and **caused** \neg *Loaded* **if not** *Loaded*. In this case, t is still an optimistic plan for*

(1)	initially caused <i>See</i> , \neg <i>Loaded</i> \rightsquigarrow $See(init) \leftarrow;$ $\neg Loaded(init) \leftarrow;$
(2)	caused <i>Loaded</i> if not \neg <i>Loaded</i> after <i>Loaded</i> \rightsquigarrow $Loaded(y) \leftarrow Loaded(x), Tr(x, y), not \neg Loaded(y)$
(3)	caused <i>See</i> if not \neg <i>See</i> after <i>See</i> \rightsquigarrow $See(y) \leftarrow See(x), Tr(x, y), not \neg See(y)$
(4)	executable <i>load</i> if \neg <i>Loaded</i> \rightsquigarrow $Exec_{load}(x) \leftarrow \neg Loaded(x)$
(5)	executable <i>shoot</i> if <i>Loaded</i> \rightsquigarrow $Exec_{shoot}(x) \leftarrow Loaded(x)$
(6)	caused <i>Loaded</i> after <i>load</i> \rightsquigarrow $Loaded(y) \leftarrow load(x, y), Tr(x, y)$
(7)	caused <i>Hit</i> after <i>See</i> , <i>shoot</i> \rightsquigarrow $Hit(y) \leftarrow See(x, y), shoot(x, y), Tr(x, y)$

Table V. Example of Planning Domain Encoding (mapping via \rightsquigarrow)

the domain, but is not secure (as the first step might not be executable). On the other hand, if hypothetically t would happen, then *Hit* would be both possibly and necessarily true after it.

To provide a procedure for deciding plan existence in the planning domains of \mathcal{K} , the authors of [Eiter et al. 2004] encode the domain into a disjunctive Datalog program and reformulate plan existence in terms of brave entailment. Since Datalog does not allow for function symbols, the encoding uses constants to instantiate the necessary successor stages. Obviously, only a finite number of constants can be used and hence, it has to be fixed in advance. For this reason the encoding is not general; only plans of certain length can be captured. Furthermore, such an encoding may also incur high space requirements.

The encoding into FDNC solves the problems from above. The availability of function symbols allows to easily generate an infinite time-line, and, hence, to avoid the usage of constants. Due to the properties of FDNC, the encoding also allows to generate the successors states “on-demand” during the model construction; in this way, space might be saved.

We remark that using higher arity FDNC, we can represent the Yale-Shooting scenario alternatively using a generic predicate $holds(f, x)$ to express truth of the fluent f in a situation x , where f is reified using a constant symbol, in the style of [Baral 2002]; e.g., $holds(Loaded, init)$ corresponds then to $Loaded(init)$. Further predicates, e.g. $abnormal(f, x)$, can be used to express other aspects of fluents. While the syntax of FDNC does not allow reification of fluents with parameters, e.g. $on(A, B)$ to $holds(on(A, B), x)$, which is also used [Baral 2002], this can be easily accommodated with tailored predicates, e.g. $holds_{on}(A, B, x)$; on the other hand, an extension of the syntax of FDNC programs that allows such terms in local positions is easily accomplished, and does not affect the worst case complexity.