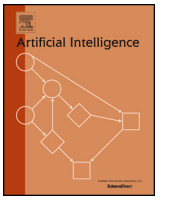
Artificial Intelligence 326 (2024) 104033

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



Syntactic ASP forgetting with forks ☆

Felicidad Aguado [a], Pedro Cabalar [a,*], Jorge Fandinno [b], David Pearce [c], Gilberto Pérez [a], Concepción Vidal [a]

[a] University of A Coruña, Spain
[b] University of Nebraska at Omaha, NE, USA
[c] Universidad Politécnica de Madrid, Spain

ARTICLE INFO

Keywords:
Answer set programming
Equilibrium logic
Forgetting
Strong persistence
Strong equivalence
Forks

ABSTRACT

Answer Set Programming (ASP) constitutes nowadays one of the most successful paradigms for practical Knowledge Representation and declarative problem solving. The formal analysis of ASP programs is essential for a rigorous treatment of specifications, the correct construction of solvers and the extension with other representational features. In this paper, we present a syntactic transformation, called the *unfolding* operator, that allows forgetting an atom in a logic program (under ASP semantics). The main advantage of unfolding is that, unlike other syntactic operators, it is always applicable and guarantees strong persistence, that is, the result preserves the same stable models with respect to any context where the forgotten atom does not occur. The price for its completeness is that the result is an expression that may contain the fork operator. Yet, we illustrate how, in some cases, the application of fork properties may allow us to reduce the fork to a logic program.

Answer Set Programming [1,2] (ASP) has become nowadays one of the most popular paradigms for practical Knowledge Representation (KR) and declarative problem solving. From the computational viewpoint, ASP tools provide a comparable performance to that of modern SAT solvers, whereas from a representational point of view, ASP offers a rich input language that allows combining defaults and non-monotonic reasoning with useful constructs such as choice rules, aggregates or optimisation specifications. Besides, ASP has been extended to cover other interesting KR features such as temporal reasoning [3] and planning [4], epistemic reasoning [5], numerical constraints [6], or updates [7], to name a few. Additionally, there exists a wide umbrella of practical application domains[1] that consolidates ASP among one of the most successful paradigms for symbolic Artificial Intelligence.

A common representational technique in ASP is the use of auxiliary atoms. Their introduction in a program may be due to many different reasons, for instance, looking for a simpler reading, providing new constructions (choice rules, aggregates, transitive closure, etc.) or reducing the corresponding ground program. When a program (or program fragment) Π for signature $\mathcal{AT}$ uses

☆ This paper is an invited revision of a paper which first appeared at the 2022 International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR-22).

* Corresponding author.
E-mail addresses: aguado@udc.es (F. Aguado), cabalar@udc.es (P. Cabalar), jfandinno@unomaha.edu (J. Fandinno), david.pearce@upm.es (D. Pearce), gilberto.pvega@udc.es (G. Pérez), concepcion.vidalm@udc.es (C. Vidal).

[1] See for instance the survey by Erdem et al. [8] published in 2016, although the number of practical applications has continually increased since then.

https://doi.org/10.1016/j.artint.2023.104033
Received 19 December 2022; Received in revised form 10 October 2023; Accepted 14 October 2023

Available online 20 October 2023

auxiliary atoms $A \subseteq \mathcal{AT}$, they do not have a relevant meaning outside Π. Accordingly, they are usually removed[2] from the final stable models, so the latter only use atoms in $V = \mathcal{AT} \setminus A$, that is, the relevant or *public vocabulary* that encodes the solutions to our problem in mind. Thus, when seen from outside, Π hides internal atoms from A and provides solutions in terms of public atoms from V. A reasonable question is whether we can reformulate some program Π exclusively in terms of public atoms V, forgetting the auxiliary ones in A. A *forgetting operator* $\mathtt{f}(\Pi, A) = \Pi'$ transforms a logic program Π into a new program Π' that does not contain atoms in A but has a *similar* behaviour on the public atoms V. Of course, the key point here is the definition of similarity between Π and Π' (relative to V) something that gave rise to different alternative forgetting operators, further classified in families, depending on the properties they satisfy – see the recent overview by Gonçalves et al. [9]. From all this wide spectrum, however, when our purpose is forgetting auxiliary atoms, similarity can only be understood as *preserving the same knowledge* for public atoms in V, and this can be formalised as a very specific property. In particular, both programs Π and $\Pi' = \mathtt{f}(\Pi, A)$ should not only produce the same stable models (projected on V) but also keep doing so even if we add a new piece of program Δ without atoms in A. This property, known as *strong persistence*, was introduced by Knorr and Alferes in 2014 [10] but, later on, Gonçalves et al. [11] proved that it is not always possible to forget A in an arbitrary program Π under strong persistence. Moreover, Gonçalves et al. also provided a semantic condition, called Ω, on the models of Π in the logic of Here-and-There (HT) [12] (the monotonic basis of *Equilibrium Logic* [13], which generalises ASP to arbitrary formulas) so that atoms A are forgettable in Π iff Ω does not hold. When this happens, their approach can be used to construct $\mathtt{f}(\Pi, A)$ from the HT models using, for instance, the methods by Cabalar et al. [14,15]. Going one step further in this model-based orientation for forgetting, Aguado et al. [16] overcame the limitation of unforgettable sets of atoms at the price of introducing a new type of disjunction, called *fork* and represented as '|'. To this aim, they defined an HT-based denotational semantics for forks. Besides, they showed a polynomial reduction from programs with forks into standard logic programs in ASP (paving the way for their direct implementation), but it requires the addition of auxiliary atoms, as could be expected.

Semantic-based forgetting is useful when we are interested in obtaining a compact representation. For instance, the method by Cabalar et al. [15] allows obtaining a minimal logic program from a set of HT-countermodels. However, this is done at a high computational cost (similar to Boolean function minimisation techniques). When combined with the Ω-condition or, similarly, with the use of HT-denotations, this method becomes practically unfeasible without the use of a computer. This may become a problem, for instance, when we try to prove properties of some new ways of using auxiliary atoms in a given setting, since one would expect a human-readable proof rather than resorting to a computer-based exhaustive exploration of models. Furthermore, semantic forgetting may easily produce results that look substantially different from the original program, even when this is not necessary. For example, if we apply an empty forgetting $\mathtt{f}(\Pi, \emptyset)$ strictly under this method, we will usually obtain a different program Π', strongly equivalent to Π, but built up from countermodels of the latter, possibly having a very different syntactic look.

An alternative and in some sense complementary orientation for forgetting is the use of *syntactic transformations*. The first syntactic forgetting operator, $\mathtt{f}_{as}$, that satisfied strong persistence was introduced by Knorr and Alferes [10]. This operator forgot a single atom $A = \{a\}$ at a time and was applicable, under some conditions, to non-disjunctive logic programs. More recently, Berthold et al. [17] presented a more general syntactic operator $\mathtt{f}_{sp}$, also for a single atom $A = \{a\}$, that can be applied to any arbitrary logic program and satisfies strong persistence when the atom can be forgotten (i.e., the Ω condition does not hold). Moreover, they also provided three syntactic sufficient conditions (that they call *a-forgettable*) under which Ω does not hold, and so, under which $\mathtt{f}_{sp}$ is strongly persistent. Perhaps the main difficulty of $\mathtt{f}_{sp}$ comes from its technically elaborate definition: it involves 10 different types of rule-matching that further deal with multiple partitions of Π (using a construction called *as-dual*). As a result, even though it offers full generality when the atom is forgettable, its application by hand does not seem very practical, requiring too many steps and a careful reading of the transformations. A second limitation of $\mathtt{f}_{sp}$ is that even if a set of atoms can be forgotten as a whole, it may not be possible to forget any single atom by itself. This limitation was recently lifted by Berthold [18], introducing a new operator $\mathtt{f}^*_{sp}$ that can be iterated. Yet, the difficulty of an elaborate definition remains in $\mathtt{f}^*_{sp}$.

In this paper, we provide a general syntactic operator, called *unfolding*, that is always applicable and allows forgetting an atom in a program, although it produces a result that may combine forks and arbitrary propositional formulas. We also discuss some examples in which a fork can be removed in favour of a formula, something that allows one to obtain a standard program (since formulas can always be reduced to that form, as proved by Cabalar et al. [19]). We show examples where sufficient syntactic conditions identified so far are not applicable, whereas our method can still safely be applied to obtain a correct result, relying on properties of forks. Unfolding relies on another syntactic operator for forgetting a single atom, $\mathtt{f}_{c}$, based on the *cut rule* from sequent calculus and is close to the application of the $\mathtt{f}_{sp}$ operator by Berthold et al. [17]. This operator produces a propositional formula without forks, but is only applicable under some sufficient syntactic conditions.

The rest of the paper is organised as follows. The next section contains the background with definitions and results from HT, stable models and the semantics of forks. After that, we present the cut transformation that produces a propositional formula. In the next section, we introduce the unfolding, which makes use of the cut and produces a fork in the general case. Finally, we conclude the paper.

This paper is an extended version of a previous publication [20] in the conference *Logic Programming and Non-Monotonic Reasoning* (LPNMR 2022). With respect to the conference paper, the main changes are: (1) an extended background section with more details, intuitions and properties of the denotational semantics of forks, used in the proofs; (2) new results included in Lemma 1, Proposition 7 and Corollary 2, the last two based on the new Definition 8; and, finally, (3) an appendix with all the proofs.

[2] Most ASP solvers allow hiding the extension of some chosen predicates.

1. Background

1.1. Here-and-there and T-supports

We begin by recalling some basic definitions and results related to the logic of HT. Let $\mathcal{AT}$ be a finite set of atoms called the *alphabet* or *vocabulary*. A *(propositional) formula* φ is defined using the grammar:

$$\varphi \ ::= \ \bot \ \Big| \ p \ \Big| \ \varphi \wedge \varphi \ \Big| \ \varphi \vee \varphi \ \Big| \ \varphi \rightarrow \varphi$$

where p is an atom $p \in \mathcal{AT}$. We define the *language* $\mathcal{L}_{\mathcal{AT}}$ as the set of all propositional formulas that can be formed over alphabet $\mathcal{AT}$. We use Greek letters φ, ψ, γ and their variants to stand for formulas. Implication $\varphi \rightarrow \psi$ will be sometimes reversed as $\psi \leftarrow \varphi$. We also define the derived operators $\neg\varphi \stackrel{\text{def}}{=} (\varphi \rightarrow \bot)$, $\top \stackrel{\text{def}}{=} \neg\bot$ and $\varphi \leftrightarrow \psi \stackrel{\text{def}}{=} (\varphi \rightarrow \psi) \wedge (\varphi \leftarrow \psi)$. Given a formula φ, by $At(\varphi) \subseteq \mathcal{AT}$ we denote the set of atoms occurring in φ. We use letters p, q, a, b for representing atoms in $\mathcal{AT}$, but normally use a for an auxiliary atom to be forgotten. A *theory* Γ is a finite[3] set of formulas that can be also understood as their conjunction. When a theory consists of a single formula $\Gamma = \{\varphi\}$ we will frequently omit the braces. Given any theory Γ, we write $\Gamma[\gamma/\varphi]$ to denote the uniform substitution of all occurrences of subformula γ in Γ by formula φ. An *extended disjunctive rule* r is an implication of the form:

$$p_1 \ \wedge \ \cdots \ \wedge \ p_m \ \wedge \ \neg p_{m+1} \ \wedge \ \cdots \ \wedge \ \neg p_n \ \wedge \ \neg\neg p_{n+1} \ \wedge \ \cdots \ \wedge \ \neg\neg p_k \quad \rightarrow \quad p_{k+1} \ \vee \ \cdots \ \vee \ p_h$$

where all p_i above are atoms in $\mathcal{AT}$ and $0 \leq m \leq n \leq k \leq h$. The antecedent and consequent of a rule r are respectively called the *body* and the *head*. We define the sets of atoms $Hd(r) \stackrel{\text{def}}{=} \{p_{k+1}, \ldots, p_h\}$, $Bd^+(r) \stackrel{\text{def}}{=} \{p_1, \ldots, p_m\}$, $Bd^-(r) \stackrel{\text{def}}{=} \{p_{m+1}, \ldots, p_n\}$, $Bd^{--}(r) \stackrel{\text{def}}{=} \{p_{n+1}, \ldots, p_k\}$ and $Bd(r) \stackrel{\text{def}}{=} Bd^+(r) \cup Bd^-(r) \cup Bd^{--}(r)$. We say that r is an *extended normal rule* if $|Hd(r)| \leq 1$. A rule with $Hd(r) = \emptyset$ is called a *constraint*. A rule with $Bd(r) = \emptyset$ and $|Hd(r)| = 1$ is called a *fact*. Given some atom a, a rule r is said to contain an *a-choice* if $a \in Bd^{--}(r) \cap Hd(r)$, that is, the rule has the form $\varphi \wedge \neg\neg a \rightarrow \psi \vee a$. A program is a finite set of rules, sometimes represented as their conjunction. We say that program Π belongs to a syntactic category if all its rules belong to that category. For instance, Π is an extended normal program if all its rules are extended normal rules. We will usually refer to the most general class, extended disjunctive logic programs, just as logic programs for short.

A *classical interpretation* T is a set of atoms $T \subseteq \mathcal{AT}$. We write $T \vDash \varphi$ to stand for the usual classical satisfaction of a formula φ. An HT-*interpretation* is a pair $\langle H, T\rangle$ (respectively called "here" and "there") of sets of atoms $H \subseteq T \subseteq \mathcal{AT}$; it is said to be *total* when $H = T$. Intuitively, this can be seen as a three-valued interpretation where an atom p can be *false*, when $p \notin T$, or *true* when $p \in T$, but for the latter, we may additionally distinguish between being *justified* (or *founded*) when $p \in H$ or *just assumed* when $p \in T \setminus H$. The fact that an interpretation $\langle H, T\rangle$ *satisfies* a formula φ, written $\langle H, T\rangle \vDash \varphi$, is recursively defined as follows:

- $\langle H, T\rangle \nvDash \bot$
- $\langle H, T\rangle \vDash p$ iff $p \in H$
- $\langle H, T\rangle \vDash \varphi \wedge \psi$ iff $\langle H, T\rangle \vDash \varphi$ and $\langle H, T\rangle \vDash \psi$
- $\langle H, T\rangle \vDash \varphi \vee \psi$ iff $\langle H, T\rangle \vDash \varphi$ or $\langle H, T\rangle \vDash \psi$
- $\langle H, T\rangle \vDash \varphi \rightarrow \psi$ iff both (i) $T \vDash \varphi \rightarrow \psi$ and (ii) $\langle H, T\rangle \nvDash \varphi$ or $\langle H, T\rangle \vDash \psi$

An HT-interpretation $\langle H, T\rangle$ is a *model* of a theory Γ if $\langle H, T\rangle \vDash \varphi$ for all $\varphi \in \Gamma$. As usual, a formula φ is a *tautology* if it is satisfied by every possible HT-interpretation, *inconsistent* if it is satisfied by no HT-interpretation, and *contingent* if it is neither a tautology nor inconsistent.

Two formulas (or theories) φ and ψ are HT-equivalent, written $\varphi \equiv \psi$, if they have the same HT-models. The logic of HT satisfies the law of substitution of logical equivalents so, in particular, if Π is a logic program, it holds that:

$$\Pi \wedge a \ \equiv \ \Pi \wedge (a \leftrightarrow \top) \ \equiv \ \Pi[a/\top] \wedge a \tag{1}$$

$$\Pi \wedge \neg a \ \equiv \ \Pi \wedge (a \leftrightarrow \bot) \ \equiv \ \Pi[a/\bot] \wedge \neg a \tag{2}$$

$$\Pi \wedge \neg\neg a \ \equiv \ \Pi \wedge (\neg a \leftrightarrow \bot) \ \equiv \ \Pi[\neg a/\bot] \wedge \neg\neg a \tag{3}$$

A total interpretation $\langle T, T\rangle$ is an *equilibrium model* of a formula φ iff $\langle T, T\rangle \vDash \varphi$ and there is no $H \subset T$ such that $\langle H, T\rangle \vDash \varphi$. If so, we say that T is a *stable model* of φ. We write $SM(\varphi)$ to stand for the set of stable models of φ and $SM_V(\varphi) \stackrel{\text{def}}{=} \{\, T \cap V \mid T \in SM(\varphi) \,\}$ for its projection onto some vocabulary V.

Aguado et al. [16] introduced a compact way of dealing with sets of HT-models by proposing the formal concept of T-*support*, defined as follows.

Definition 1. Given $T \subseteq \mathcal{AT}$, a T-*support* $\mathcal{H}$ is a set of subsets of T, that is $\mathcal{H} \subseteq 2^T$, satisfying that $T \in \mathcal{H}$ if $\mathcal{H} \neq \emptyset$.

[3] As we will see, the cut operator support is a conjunction built from a finite set of rules that is sometimes negated. Generalising to infinite theories would require infinitary Boolean connectives.

The intuition behind a T-support is that it will be used to collect those H such that $\langle H, T\rangle$ are models of a given formula φ. To increase readability, we write a support as a sequence of interpretations between square braces. For instance, some possible supports for $T = \{a, b\}$ are $[\{a, b\}\ \{a\}]$, $[\{a, b\}\ \{b\}\ \emptyset]$ or the empty support $[\ ]$. Given a propositional formula φ and $T \subseteq \mathcal{AT}$, the set of HT-models $\{H \subseteq T \mid \langle H, T\rangle \models \varphi\}$ forms a T-support we denote as $[\![\varphi]\!]^T$.

For any T-support $\mathcal{H}$ and set of atoms V, we write $\mathcal{H}_V$ to stand for $\{H \cap V \mid H \in \mathcal{H}\}$.

We say that a T-support $\mathcal{H}$ is *V-feasible* iff there is no $H \subset T$ in $\mathcal{H}$ satisfying that $H \cap V = T \cap V$. The name comes from the fact that, if this condition does not hold for some $\mathcal{H} = [\![\varphi]\!]^T$ with $H \subset T$, then T cannot be stable for any formula $\varphi \wedge \psi$ with $\psi \in \mathcal{L}_V$ because H and T are indistinguishable for any formula $\psi \in \mathcal{L}_V$.

We can define an ordering relation $\preceq$ between T-supports by saying that, given two T-supports, $\mathcal{H}$ and $\mathcal{H}'$, $\mathcal{H} \preceq \mathcal{H}'$ iff either $\mathcal{H} = [\ ]$ or $[\ ] \neq \mathcal{H}' \subseteq \mathcal{H}$. The intuition of $\mathcal{H} \preceq \mathcal{H}'$ is that $\mathcal{H}'$ is "more supported" than $\mathcal{H}$ in the sense that, the less elements in a (non-empty) support, the closer we are to produce T as a stable model of some formula. The most supported T-support is therefore $[T]$, that constitutes the top element of the $\preceq$ relation: when $[\![\varphi]\!]^T = [T]$, it means that T is, indeed, a stable model of φ. The bottom element of $\preceq$ is the empty support $[\ ]$ so that $[\![\varphi]\!]^T = [\ ]$ is considered to be the case furthest away from getting T as stable model, since T is not even a classical model of φ.

Given a T-support $\mathcal{H}$, we define its complementary support $\overline{\mathcal{H}}$ as:

$$\overline{\mathcal{H}} \stackrel{\text{def}}{=} \begin{cases} [\ ] & \text{if } \mathcal{H} = 2^T \\ [\ T\ ] \cup \{H \subseteq T \mid H \notin \mathcal{H}\} & \text{otherwise.} \end{cases}$$

1.2. Overview of forks

Aguado et al. [16,21] extended logic programs to include a new disjunctive construct '|' that can be combined with other connectives under a limited syntax. A *fork* F is an expression determined by the following grammar:

$$F \ ::= \ \bot \ \Big| \ p \ \Big| \ (F \mid F) \ \Big| \ F \wedge F \ \Big| \ \varphi \vee \varphi \ \Big| \ \varphi \rightarrow F$$

where φ is a propositional formula over $\mathcal{AT}$ and $p \in \mathcal{AT}$ is an atom. We write $\mathcal{L}_{\mathcal{AT}}$ to stand for the language formed by all forks for signature $\mathcal{AT}$. Note that a fork is not allowed as an argument of a disjunction or as the antecedent of an implication. Given a fork $(F \mid G)$, we say that F and G are its *branches*, respectively.

To introduce the semantics of forks, consider a fork F of the form $(\varphi_1 \mid \cdots \mid \varphi_n)$ (in fact, any fork will be reducible to this form) where each φ_i is a propositional formula or, if preferred, a logic program. The intuition about the stable models of F is to collect the *union* of the stable models of each φ_i. Therefore, for a fixed T, the semantics for F could just be a set of T-supports $\Delta = \{\mathcal{H}_1, \ldots, \mathcal{H}_n\}$ corresponding to the respective denotations $[\![\varphi_1]\!]^T, \ldots, [\![\varphi_n]\!]^T$. However, if we have some $\mathcal{H}_i \preceq \mathcal{H}_j$ for $i \neq j$, then $\mathcal{H}_i$ becomes useless in the sense that any stable model produced by $\mathcal{H}_i$ is also produced by $\mathcal{H}_j$. For this reason, rather than considering a set of supports, we will use their *ideals* so that the relevant information comes from the set of maximal elements in the set. The *ideal* of $\mathcal{H}$ is defined as $\downarrow\mathcal{H} = \{\mathcal{H}' \mid \mathcal{H}' \preceq \mathcal{H}\} \setminus \{[\ ]\}$. Note that, the empty support $[\ ]$ is not included in the ideal, so $\downarrow[\ ] = \emptyset$. If Δ is any set of supports:

$$^{\downarrow}\Delta \stackrel{\text{def}}{=} \bigcup_{\mathcal{H} \in \Delta} \downarrow\mathcal{H} = \bigcup_{\mathcal{H} \in \Delta} \{\ \mathcal{H}' \preceq \mathcal{H} \ \Big|\ \mathcal{H}' \neq [\ ]\ \}.$$

We formalise the previous ideas with the following definition of *T-view*, the semantic structure we will associate to a fork.

Definition 2. A *T-view* Δ is a set of T-supports that is $\preceq$-closed, i.e., $^{\downarrow}\Delta = \Delta$.

We provide next the semantics of forks in terms of *T-denotations*. To illustrate the duality in the definitions of conjunction and disjunction, we will use a weaker version of the membership relation, $\hat{\in}$, defined as follows. Given a T-view Δ, we write $\mathcal{H} \hat{\in} \Delta$ iff $\mathcal{H} \in \Delta$ or both $\mathcal{H} = [\ ]$ and $\Delta = \emptyset$.

Definition 3 *(T-denotation).* Let $\mathcal{AT}$ be a propositional signature and $T \subseteq \mathcal{AT}$ a set of atoms. The *T-denotation* of a fork or a propositional formula F, written $\langle\!\langle F \rangle\!\rangle^T$, is a T-view recursively defined as follows:

$$\begin{aligned}
\langle\!\langle \bot \rangle\!\rangle^T &\stackrel{\text{def}}{=} \emptyset \\
\langle\!\langle p \rangle\!\rangle^T &\stackrel{\text{def}}{=} \downarrow [\![p]\!]^T \quad \text{for any atom } p \\
\langle\!\langle F \wedge G \rangle\!\rangle^T &\stackrel{\text{def}}{=} {}^{\downarrow}\{\ \mathcal{H} \cap \mathcal{H}' \ \Big|\ \mathcal{H} \in \langle\!\langle F \rangle\!\rangle^T \text{ and } \mathcal{H}' \in \langle\!\langle G \rangle\!\rangle^T\ \} \\
\langle\!\langle \varphi \vee \psi \rangle\!\rangle^T &\stackrel{\text{def}}{=} {}^{\downarrow}\{\ \mathcal{H} \cup \mathcal{H}' \ \Big|\ \mathcal{H} \hat{\in} \langle\!\langle \varphi \rangle\!\rangle^T \text{ and } \mathcal{H}' \hat{\in} \langle\!\langle \psi \rangle\!\rangle^T\ \} \\
\langle\!\langle \varphi \rightarrow F \rangle\!\rangle^T &\stackrel{\text{def}}{=} \begin{cases} \{2^T\} & \text{if } [\![\varphi]\!]^T = [\ ] \\ {}^{\downarrow}\{\ \overline{[\![\varphi]\!]^T} \cup \mathcal{H} \ \Big|\ \mathcal{H} \in \langle\!\langle e\, F \rangle\!\rangle^T\ \} & \text{otherwise} \end{cases} \\
\langle\!\langle F \mid G \rangle\!\rangle^T &\stackrel{\text{def}}{=} \langle\!\langle F \rangle\!\rangle^T \cup \langle\!\langle G \rangle\!\rangle^T
\end{aligned}$$

where F, G denote forks or propositional formulas. If F is a fork and $T \subseteq V \subseteq \mathcal{AT}$, we can define the T-view:

$$\langle\!\langle\, F \,\rangle\!\rangle_V^T \stackrel{\text{def}}{=} \downarrow\{\, \mathcal{H}_V \;\big|\; \mathcal{H} \in \langle\!\langle\, F \,\rangle\!\rangle^Z \text{ s.t. } Z \cap V = T \text{ and } \mathcal{H} \text{ is } V\text{-feasible} \,\}.$$

Given fork F for vocabulary $\mathcal{AT}$ we say that $T \subseteq \mathcal{AT}$ is a *stable model* of F when $\langle\!\langle\, F \,\rangle\!\rangle^T = \downarrow[\, T \,]$ or, equivalently, when $[\, T \,] \in \langle\!\langle\, F \,\rangle\!\rangle^T$. The set $SM(F)$ collects all the stable models of F whereas $SM_V(F)$ denotes the projection of $SM(F)$ on the set of atoms V, that is $SM_V(F) \stackrel{\text{def}}{=} \{T \cap V \mid T \in SM(F)\}$.

Definition 4 *(Projective Strong Entailment/Equivalence of forks).* Let F and G be forks and $V \subseteq \mathcal{AT}$ a set of atoms. We say that F *strongly V-entails* G, in symbols $F \mid\!\sim_V G$, if for any fork L in $\mathcal{L}_V$, $SM_V(F \wedge L) \subseteq SM_V(G \wedge L)$. We further say that F and G *are strongly V-equivalent*, in symbols $F \cong_V G$ if both $F \mid\!\sim_V G$ and $G \mid\!\sim_V F$, that is, $SM_V(F \wedge L) = SM_V(G \wedge L)$, for any fork L in $\mathcal{L}_V$. When $\mathcal{AT}(F) \cup \mathcal{AT}(G) \subseteq V$, we write $F \mid\!\sim G$ (or $F \cong G$) dropping the V sub-index and simply saying that F *strongly entails* G (or F and G are *strongly equivalent*).

A *forgetting operator* is a transformation $\mathtt{f}(F, A)$ that takes some expression F (a fork or a formula) for alphabet $\mathcal{AT}$ as an input and produces a new expression that only contains atoms in $V = \mathcal{AT} \setminus A$. When $A = \{a\}$ is a singleton, we normally write $\mathtt{f}(F, a)$ instead of $\mathtt{f}(F, \{a\})$. A forgetting operator $\mathtt{f}(F, A)$ satisfies *strong persistence* when $\varphi \cong_V \mathtt{f}(\varphi, A)$ where $V = \mathcal{AT} \setminus A$.

The properties listed in the following theorem were proved by Aguado et al. [16].

Theorem 1. *Let F and G be arbitrary forks, and φ and ψ propositional formulas all of them for signature $\mathcal{AT}$, and let $V \subseteq \mathcal{AT}$. Then:*

(i) $F \cong_V G$ *iff* $\langle\!\langle\, F \,\rangle\!\rangle_V^T = \langle\!\langle\, G \,\rangle\!\rangle_V^T$, *for every* $T \subseteq V$.
(ii) $F \cong G$ *iff* $\langle\!\langle\, F \,\rangle\!\rangle^T = \langle\!\langle\, G \,\rangle\!\rangle^T$, *for every* $T \subseteq \mathcal{AT}$.
(iii) $\langle\!\langle\, \varphi \,\rangle\!\rangle^T = \downarrow[\![\, \varphi \,]\!]^T$ *for every* $T \subseteq \mathcal{AT}$.
(iv) $\varphi \cong \psi$ *iff* $[\![\, \varphi \,]\!]^T = [\![\, \psi \,]\!]^T$, *for every* $T \subseteq \mathcal{AT}$, *iff* $\varphi \equiv \psi$ *in HT.*
(v) The set of atoms $\mathcal{AT} \setminus V$ *can be forgotten in* F *as a strongly persistent propositional formula*[4] *iff for each* $T \subseteq V$, $\langle\!\langle\, F \,\rangle\!\rangle_V^T$ *has a unique maximal support.* □

When $\langle\!\langle\, F \,\rangle\!\rangle^T \subseteq \langle\!\langle\, G \,\rangle\!\rangle^T$ for every $T \subseteq \mathcal{AT}$ we say that F *strongly entails* G, written $F \mid\!\sim G$. For propositional formulas, we get the following characterisation of strong entailment.

Proposition 1 *(From Proposition 3 in [16]). Given propositional formulas φ and ψ, $\varphi \mid\!\sim \psi$ iff both: (1) φ classically entails ψ; and (2), $\langle H, T\rangle \models \varphi$ for any $H \subseteq T$ such that $\langle H, T\rangle \models \psi$ and $T \models \varphi$.*

Relation $\mid\!\sim$ can be seen as one of the two sides of strong equivalence: for instance, $(\neg p \to q) \mid\!\sim (p \vee q)$ holds and it means that any stable model of $\Pi \cup \{\neg p \to q\}$ is also a stable model of $\Pi \cup \{p \vee q\}$ for any context program Π. It must be noted that, in general, relations $\varphi \mid\!\sim \psi$ (strong entailment) and $\varphi \models \psi$ (HT entailment) do not coincide: in fact $(\neg p \to q) \not\mid\!\sim (p \vee q)$ since $\langle \emptyset, \{p\}\rangle$ is an HT-model of $\neg p \to q$ but not of $p \vee q$.

We can semantically characterise propositional formulas as T-denotations that have a $\preceq$-maximum element:

Proposition 2 *(Proposition 17 in [16]). Given sets $T \subseteq V \subseteq \mathcal{AT}$ of atoms, then:*

(i) *any formula φ with $\mathcal{AT}(\varphi) \subseteq V$ satisfies $\langle\!\langle\, \varphi \,\rangle\!\rangle_V^T = \downarrow[\![\, \varphi \,]\!]^T$ and, thus, $\langle\!\langle\, \varphi \,\rangle\!\rangle_V^T$ has a $\preceq$-maximum element;*
(ii) *for every T-view Δ with a $\preceq$-maximum element, there is a propositional formula φ with $\mathcal{AT}(\varphi) \subseteq V$ that satisfies $\langle\!\langle\, \varphi \,\rangle\!\rangle_V^T = \Delta$ and $\langle\!\langle\, \varphi \,\rangle\!\rangle_V^{T'} = \emptyset$ for every $T' \subseteq V$ with $T' \neq T$.*

The next propositions contain some equivalences we will use later on.

Proposition 3. *Let F, F', G and G' be forks for some signature $\mathcal{AT}$ and let $V \subseteq \mathcal{AT}$. If $F \cong_V F'$ and $G \cong_V G'$, then $(F \mid G) \cong_V (F' \mid G')$.* □

Proposition 4 *(Proposition 12 in [16]). Let F, G, L be arbitrary forks and φ be a formula. Then:*

$$(F \mid G) \mid L \cong F \mid (G \mid L) \tag{4}$$

$$F \mid G \cong G \mid F \tag{5}$$

$$(F \mid G) \cong G \qquad \text{if } F \mid\!\sim G \tag{6}$$

$$(F \mid G) \wedge L \cong (F \wedge L) \mid (G \wedge L) \tag{7}$$

[4] This is, therefore, equivalent to not satisfying the Ω condition by Gonçalves et al. [11].

$$\varphi \to (F \mid G) \cong (\varphi \to F) \mid (\varphi \to G) \tag{8}$$

$$\varphi \to F \wedge G \cong (\varphi \to F) \wedge (\varphi \to G) \tag{9}$$

$$\varphi \to (\psi \to F) \cong \varphi \wedge \psi \to F \tag{10}$$

$$\top \to F \cong F \tag{11}$$

$$\neg\varphi \mid \neg\neg\varphi \cong \top \tag{12}$$

Note how (6) is a subsumption property that guarantees that $(F \mid G) \cong G$ when F strongly entails G. This is somehow analogous to the property of disjunction in classical logic $(F \vee G) \equiv G$ when $F \models G$.

Proposition 5. *For every pair φ, ψ of propositional formulas and fork F:*

$$(\top \mid \varphi) \cong (\neg\varphi \mid \varphi) \cong \neg\neg\varphi \to \varphi \cong \varphi \vee \neg\varphi \tag{13}$$

$$(\varphi \wedge \neg\psi \mid \varphi \wedge \neg\neg\psi) \cong \varphi \tag{14}$$

$$(\bot \mid F) \cong F \tag{15}$$

2. The cut operator

Given any program Π, let us define the syntactic transformation $behead^a(\Pi)$ as the result of removing all rules with $a \in Hd(r) \cap Bd^+(r)$ and all head occurrences of a from rules where $a \in Hd(r) \cap Bd^-(r)$. Intuitively, $behead^a(\Pi)$ removes from Π all rules that, having a in the head, do not provide a support for a. In fact, rules with $a \in Hd(r) \cap Bd^+(r)$ are tautological, whereas rules of the form $\varphi \wedge \neg a \to a \vee \psi$ are strongly equivalent to $\varphi \wedge \neg a \to \psi$. Since the logic program transformations in $behead^a(\Pi)$ are strongly equivalent, we can easily see that:

Proposition 6. *For any logic program Π: $\Pi \cong behead^a(\Pi)$.* □

The cut operator is defined in terms of the well-known *cut inference rule* from the sequent calculus which, when rephrased for program rules, amounts to:

$$\frac{\varphi \wedge a \to \psi \qquad \varphi' \to a \vee \psi'}{\varphi \wedge \varphi' \to \psi \vee \psi'} \tag{CUT}$$

where φ, φ' are conjunctions of elements that can be an atom p, its negation $\neg p$ or its double negation $\neg\neg p$, and ψ' and ψ are disjunctions of atoms. If r and r' stand for $\varphi \wedge a \to \psi$ and $\varphi' \to a \vee \psi'$ respectively, then we denote $Cut(a, r, r')$ to stand for the resulting implication $\varphi \wedge \varphi' \to \psi \vee \psi'$.

Example 1 *(Example 9 in [17]).* Let Π_1 be the program:

$$a \to t \tag{16}$$

$$\neg a \to v \tag{17}$$

$$s \to a \tag{18}$$

$$r \to a \vee u \tag{19}$$

Then, $Cut(a, (16), (19)) = (r \to t \vee u)$ is the result of the cut application:

$$\frac{\top \wedge a \to t \qquad r \to a \vee u}{\top \wedge r \to t \vee u}$$

In this program we can also perform a second cut through atom a corresponding to $Cut(a, (16), (18)) = (s \to t)$. □

Given a rule r with $a \in Bd^+(r)$, we define the formula:

$$NES(\Pi, a, r) \stackrel{\text{def}}{=} \bigwedge \{\ Cut(a, r, r') \mid r' \in \Pi,\ a \in Hd(r')\ \}$$

that is, $NES(\Pi, a, r)$ collects the conjunction of all possible cuts in Π for a given atom a and a selected rule r with a in the positive body. We used the acronym NES standing for "Negative External Support" due to its connection to the so-called *external support* by Ferraris et al. [22] that we will see later on. But in fact, NES coincides with the transformation described in the *General Principle of Partial Evaluation* already stated by Brass and Dix [23]. In our example program Π_1 for rule (16) we get:

$$NES(\Pi_1, a, (16)) = (r \to t \vee u) \wedge (s \to t). \tag{20}$$

When $r = \neg a = (\top \wedge a \rightarrow \bot)$ we can observe that:

$$\begin{aligned} NES(\Pi, a, \neg a) &= \bigwedge \{(\top \wedge \varphi' \rightarrow \bot \vee \psi') \mid (\varphi' \rightarrow a \vee \psi') \in \Pi\} \\ &= \bigwedge \{(\varphi' \rightarrow \psi') \mid (\varphi' \rightarrow a \vee \psi') \in \Pi\}. \end{aligned}$$

That is, we just take the rules with a in the head, but after removing a from that head. As an example, we have

$$NES(\Pi_1, a, \neg a) \;=\; (s \rightarrow \bot) \wedge (r \rightarrow u) \;=\; \neg s \wedge (r \rightarrow u).$$

Note that, since a was the only head atom in (18), after removing it, we obtained an empty head $\bot$ leading to $(s \rightarrow \bot)$.

As said before, the negation of NES can be connected with the external support by Ferraris et al. [22]. In particular, we can use de Morgan and the HT equivalence $\neg(\varphi' \rightarrow \psi') \equiv \neg\neg\varphi' \wedge \neg\psi'$ to conclude:

$$\neg NES(\Pi, a, \neg a) = \neg\neg \bigvee \{(\varphi' \wedge \neg\psi') \mid (\varphi' \rightarrow a \vee \psi') \in \Pi\} \;=\; \neg\neg ES_\Pi(a),$$

where $ES_\Pi(a)$ corresponds to the external support[5] $ES_\Pi(Y)$ by Ferraris et al. [22] for any set of atoms Y, but applied here to $Y = \{a\}$. In the example:

$$\neg NES(\Pi_1, a, \neg a) = \neg(\neg s \wedge (r \rightarrow u)) \equiv \neg\neg s \vee (\neg\neg r \wedge \neg u). \tag{21}$$

Definition 5 *(Cut operator $\mathtt{f}_c$).* Let Π be a logic program for alphabet $\mathcal{AT}$ and let $a \in \mathcal{AT}$. Then $\mathtt{f}_c(\Pi, a)$ is defined as the result of:

(i) Remove atom 'a' from non-supporting heads obtaining $\Pi' = behead^a(\Pi)$;
(ii) Replace each rule $r \in \Pi'$ with $a \in B^+(r)$ by $NES(\Pi', a, r)$;
(iii) From the result, remove every rule r with $Hd(r) = \{a\}$;
(iv) Finally, replace remaining occurrences of 'a' by $\neg NES(\Pi', a, \neg a)$. □

Looking at Definition 5 above, note that rules where a does not occur are left untouched, so if a does not occur in Π, $\mathtt{f}_c(\Pi, a) = \Pi$. On the other hand, note that a precondition for the application of $\mathtt{f}_c$ is that a does not occur in $NES(\Pi', a, \neg a)$. Otherwise, step (ii) may leave occurrences of atom a in the result.

Example 2 *(Example 1 continued).* Step (i) has no effect, since $behead^a(\Pi_1) = \Pi_1$. For step (ii), the only rule with a in the positive body is (16) and so, the latter is replaced by (20). Step (iii) removes rule (18) and, finally, Step (iv) replaces a by (21) in rules (17) and (19). Finally, $\mathtt{f}_c(\Pi_1, a)$ becomes the conjunction of:

$$(s \rightarrow t) \wedge (r \rightarrow t \vee u) \tag{22}$$

$$\neg(\neg\neg s \vee (\neg\neg r \wedge \neg u)) \rightarrow v \tag{23}$$

$$r \rightarrow \neg\neg s \vee (\neg\neg r \wedge \neg u) \vee u \tag{24}$$

Now, by simple HT transformations [19], it is easy to see that the antecedent of (23) amounts to $\neg s \wedge (\neg r \vee \neg\neg u)$, so (23) can be replaced by the two rules (25) and (26) below, whereas (24) is equivalent to the conjunction of (27) below that stems from $r \rightarrow \neg\neg s \vee \neg u \vee u$, plus the rule $r \rightarrow \neg\neg s \vee \neg\neg r \vee u$ that is tautological and can be removed.

$$\neg s \wedge \neg r \rightarrow v \tag{25}$$

$$\neg s \wedge \neg\neg u \rightarrow v \tag{26}$$

$$r \wedge \neg s \wedge \neg\neg u \rightarrow u \tag{27}$$

To sum up, $\mathtt{f}_c(\Pi_1, a)$ is strongly $\{r, s, t, u, v\}$-equivalent to program (22) ∧ (25) ∧ (26) ∧ (27). □

The program we obtained above is the same one obtained with the $\mathtt{f}_{sp}$ operator by Berthold et al. [17] although the process to achieve it is slightly different. This is because, in general, $\mathtt{f}_c(\Pi, a)$ takes a logic program Π but produces a *propositional formula* where a has been forgotten, whereas $\mathtt{f}_{sp}$ produces the logic program in a direct way. Although, at a first sight, this could be seen as a limitation of $\mathtt{f}_c$, the truth is that it is not an important restriction, since there exist well-known syntactic methods [19,25] to transform a propositional formula[6] into a (strongly equivalent) logic program under the logic of HT. Moreover, in the case of $\mathtt{f}_{sp}$, directly producing a logic program comes with the cost of a more technically elaborate transformation, with ten different cases

[5] In fact, Aguado et al. [24] presented a more limited forgetting operator $\mathtt{f}_{es}$ based on the external support.

[6] In most cases, after unfolding $\mathtt{f}_c$ as a logic program, we usually obtain not only a result strongly equivalent to $\mathtt{f}_{sp}$ but also the same or a very close syntactic representation.

and the combinatorial construction of a so-called *as-dual* set of rules generated from multiple partitions of the original program.[7] We suggest that well-known logical rules such as de Morgan or distributivity (many of them still valid in intuitionistic logic) are far easier to learn and apply than the $\mathtt{f}_{sp}$ transformation when performing syntactic transformations by hand. On the other hand, we may sometimes be interested in keeping the propositional formula representation inside HT (for instance, for studying strong equivalence or the relation to other constructions) rather than being forced to unfold the formula into a logic program, possibly leading to a combinatorial blow-up due to distributivity.

As happened with $\mathtt{f}_{sp}$, the main restriction of $\mathtt{f}_{c}$ is that it does not always guarantee strong persistence. Note that this was expected, given the already commented result on the impossibility of arbitrary forgetting by just producing an HT formula. To check whether forgetting a in Π is possible, we can use semantic conditions like Theorem 1(v) or the Ω-condition, but these imply inspecting the models of Π. If we want to keep the method at a purely syntactic level, however, we can at best enumerate sufficient conditions for forgettability. For instance, Berthold et al. [17] proved that a can be forgotten under strong persistence in any program Π that satisfies any of the following syntactic conditions:

Definition 6 *(Definition 4 in [17]).* An extended logic program Π is *a-forget-table* if at least one of the following conditions is satisfied:

1. Π contains the fact 'a' as a rule.
2. Π does not contain a-choices.
3. All rules in Π in which a occurs are a-choices.

It is not difficult to see that Condition 2 above is equivalent to requiring that atom a does not occur in $NES(\Pi, a, \neg a)$, since the only possibility for a to occur in that formula is that there is a rule in Π of the form $\neg\neg a \wedge \varphi \rightarrow a \vee \psi$. In fact, as we prove below, Definition 6 is a quite general, sufficient syntactic condition for the applicability of $\mathtt{f}_{c}$.

Theorem 2. *Let Π be a logic program for signature $\mathcal{AT}$, let $V \subseteq \mathcal{AT}$ and $a \in \mathcal{AT} \setminus V$. If Π is a-forgettable, then: $\Pi \cong_V \mathtt{f}_c(\Pi, a)$.* □

In our example, it is easy to see that this condition is satisfied because $behead^a(\Pi_1) = \Pi_1$ and this program does not contain a-choices.

3. Forgetting into forks: the *unfolding* operator

As we have seen, syntactic forgetting is limited to a family of transformation operators whose applicability can be analysed in terms of sufficient syntactic conditions. This method is incomplete in the sense that forgetting a in Π may be possible, but still the syntactic conditions we use for applicability may not be satisfied. Consider the following example.

Example 3. Take the following logic program Π_3:

$$\neg\neg a \rightarrow a \tag{28}$$

$$\neg a \rightarrow b \tag{29}$$

$$a \rightarrow c \tag{30}$$

$$b \rightarrow c \tag{31}$$

$$c \rightarrow b \tag{32}$$

This program does not fit into the a-forgettable syntactic form, but in fact we can forget a under strong persistence to obtain $b \wedge c$, as we will see later. □

If we look for a complete forgetting method, one interesting possibility is allowing the result to contain the fork operator. As proved by Aguado et al. [16], forgettability as a fork is always guaranteed: that is, it is always possible to forget any atom if we allow the result to be in the general form of a fork. The method they provided to obtain such a fork, however, was based on synthesis from the fork denotation, which deals with sets of HT models. We propose next a syntactic method, that is always applicable, to obtain a fork as the result of forgetting any atom.

In the context of propositional logic, forgetting an atom a in a formula φ corresponds to the quantified Boolean formula $\exists a\ \varphi$ which, in turn, is equivalent to the unfolding $\varphi[a/\bot] \vee \varphi[a/\top]$. In the case of Equilibrium Logic, we will apply a similar unfolding but, instead of disjunction, we will use the fork connective, and rather than $\bot$ and $\top$ we will have to divide the cases into $\neg a$ and $\neg\neg a$, since $(\neg a \mid \neg\neg a) \equiv \top$.

[7] In fact, the as-dual set defined by Berthold et al. [17] can be seen as an effect of the (CUT) rule. Moreover, our use of the latter was inspired by this as-dual construction.

More precisely, using (14) from Proposition 5 we can say that $\Pi \cong (\Pi \wedge \neg a \mid \Pi \wedge \neg\neg a)$. Then, by Proposition 3, we separate the task of forgetting a in Π into forgetting a in each one of these two branches, leading to:

Definition 7 *(Unfolding operator, $\mathtt{f}_{|}$).* For any logic program Π and atom a we define: $\mathtt{f}_{|}(\Pi, a) \stackrel{\text{def}}{=} \left(\mathtt{f}_c(\Pi \wedge \neg a, a) \mid \mathtt{f}_c(\Pi \wedge \neg\neg a, a) \right)$ □

As a minor remark, note that when we forget atom a on a program Π where a does not occur, $NES(\Pi, a, \neg a)$ becomes the empty conjunction $\top$ and we trivially obtain $(\Pi \wedge \neg\top \mid \Pi \wedge \neg\neg\top) \cong (\bot \mid \Pi) \cong \Pi$. The following result guarantees that the unfolding operator always produces a strongly persistent forgetting of atom a.

Theorem 3. *Let Π be a logic program for signature $\mathcal{AT}$, let $V \subseteq \mathcal{AT}$ and $a \in \mathcal{AT} \setminus V$. Then, $\Pi \cong_V \mathtt{f}_{|}(\Pi, a)$.* □

In general, $\mathtt{f}_{|}(\Pi, a)$ may contain forks, but when the program is a-forgettable, we immediately conclude that the cut operator $\mathtt{f}_c(\Pi, a)$ (which produces a formula) yields a strongly equivalent result:

Corollary 1. *Let Π be a logic program for signature $\mathcal{AT}$, let $V \subseteq \mathcal{AT}$ and $a \in \mathcal{AT} \setminus V$. If Π is a-forgettable, then $\mathtt{f}_{|}(\Pi, a) \cong_V \mathtt{f}_c(\Pi, a)$, and so, $\mathtt{f}_{|}(\Pi, a) \cong \mathtt{f}_c(\Pi, a)$.* □

Using (2) and (3), it is easy to prove:

Theorem 4. *For any logic program Π and atom a:*

$$\begin{aligned}\mathtt{f}_{|}(\Pi, a) &\cong \left(\mathtt{f}_c(\Pi[a/\bot] \wedge \neg a, a) \mid \mathtt{f}_c(\Pi[\neg a/\bot] \wedge \neg\neg a, a) \right) \\ &\cong \left(\Pi[a/\bot] \mid \mathtt{f}_c(\Pi[\neg a/\bot] \wedge \neg\neg a, a) \right).\end{aligned}$$

This theorem provides a simpler application of the unfolding operator: the left branch, for instance, is now the result of replacing a by $\bot$. The right branch applies the cut operator, but introducing a prior step: we add the formula $\neg\neg a$ and replace all occurrences of $\neg a$ by $\bot$. It is easy to see that, in this previous step, any occurrence of a in the scope of negation is removed in favour of truth constants.[8] This means that the result has no a-choices since a will only occur in the scope of negation in the rule $\neg\neg a = (\neg a \rightarrow \bot)$. Therefore, the use of $\mathtt{f}_c$ in $\mathtt{f}_{|}$ is always applicable. Moreover, in many cases, we can use elementary HT transformations to simplify the programs $\Pi[a/\bot]$ and $\Pi[\neg a/\bot] \wedge \neg\neg a$, to look for a simpler application of $\mathtt{f}_c$, or to apply properties about the obtained fork.
As an illustration, consider the following variation of Example 3.

Example 4. Suppose we want to forget atom a in the program $\Pi_4 \stackrel{\text{def}}{=} (28) \wedge (29) \wedge (30)$ where we simply removed (31) and (32) from Π_3.

Let us use the transformation in Theorem 4. We can observe that $\Pi_4[a/\bot]$ replaces (28), (29) and (30) respectively by $(\neg\neg\bot \rightarrow \bot)$ (a tautology), $(\neg\bot \rightarrow b) \cong b$ and $(\bot \rightarrow c)$ (again, a tautology). On the other hand, $\Pi_4[\neg a/\bot]$ replaces (28) and (29) respectively by $(\neg\bot \rightarrow a) \cong a$ and $(\bot \rightarrow b)$ (a tautology), so that $\Pi_4[\neg a/\bot] \wedge \neg\neg a$ amounts to the formula $a \wedge (a \rightarrow c) \wedge \neg\neg a$ which is equivalent to $a \wedge c$. Therefore, we get $\mathtt{f}_c(a \wedge c, a) = c$. The final result amounts to $\mathtt{f}_{|}(\Pi_4, a) = (b \mid c)$ that is, a fork of two atoms, which as discussed by Aguado et al. [16], is (possibly the simplest case of) a fork that *cannot be reduced to a formula*. Later on, we will provide, in fact, a more general sufficient condition for non-reducibility that includes this case.

Now, think again about the larger program $\Pi_3 = \Pi_4 \cup \{(31), (32)\}$ that was not a-forgettable under the sufficient syntactic conditions for that class of programs. We can indeed reuse the forgetting $\mathtt{f}_{|}(\Pi_4, a)$ to obtain $\mathtt{f}_{|}(\Pi_3, a)$ due to the following lemma:

Lemma 1. *Let Π, Π' be two logic programs for some signature $\mathcal{AT}$. Suppose that, for some atom $a \in \mathcal{AT}$, $a \notin At(\Pi')$. Then:*

$$\mathtt{f}_{|}(\Pi \cup \Pi', a) = \mathtt{f}_{|}(\Pi, a) \wedge \Pi'.$$

Since rules (31) and (32) do not contain atom a, we can apply Lemma 1 so:

$$\begin{aligned}\mathtt{f}_{|}(\Pi_3, a) &= \mathtt{f}_{|}(\Pi_4, a) \wedge (b \rightarrow c) \wedge (c \rightarrow b) \\ &\cong (b \mid c) \wedge (b \rightarrow c) \wedge (c \rightarrow b) \\ &\cong b \wedge (b \rightarrow c) \wedge (c \rightarrow b) \mid c \wedge (b \rightarrow c) \wedge (c \rightarrow b) \quad \text{by (7)} \\ &\cong b \wedge c \mid b \wedge c \\ &\cong b \wedge c. \quad \text{by (6)}\end{aligned}$$

[8] Truth constants can be removed using trivial HT simplifications.

In this way, we have *syntactically* proved that a was indeed forgettable in Π_3 leading to $b \wedge c$ even though this program was not a-forgettable. We claim that the $\mathtt{f}_|$ operator plus the use of properties about forks (like idempotence used above) opens a wider range of syntactic conditions under which forks can be reduced to formulas, and so under which an atom can be forgotten in ASP.

An important advantage of the unfolding operator is that, since it is always applicable, it can be used to forget a set of atoms by forgetting them one by one. We illustrate this with another example.

Example 5. We have shown before that $\mathtt{f}_|(\Pi_3, a) \equiv (b \mid c)$. We can use Proposition 3 to continue forgetting b in each of the two branches of $(b \mid c)$.

As none of them contains b-choices, we can just apply $\mathtt{f}_c$ to obtain the fork $(\mathtt{f}_c(b,b) \mid \mathtt{f}_c(b,c)) = (\top \mid c)$ which, by (13), is equivalent to the formula $(\neg\neg c \to c)$.

We discuss next an example extracted from the paper by Berthold et al. [17].

Example 6. Suppose we want to forget q in the following program Π_6:

$$\neg\neg q \to q \qquad q \to u \qquad q \to s \qquad \neg q \to t.$$

Although this program is not q-forgettable, it was included as Example 7 in Berthold et al.'s paper [17] to illustrate the application of operator $\mathtt{f}_{sp}$. If we use $\mathtt{f}_|(\Pi_6, q)$, it is very easy to see that $\Pi_6[q/\bot] \cong t$ and $\Pi_6[\neg q/\bot] \wedge \neg\neg q \cong q \wedge u \wedge s$. Therefore, we get $\mathtt{f}_|(\Pi_6, q) = (t \mid \mathtt{f}_c(q \wedge u \wedge s, q)) = (t \mid u \wedge s)$. As we will see below, this fork *cannot* be represented as a formula: in some sense, it is similar to $(b \mid c)$ obtained before. As a result, atom q cannot be forgotten in Π_6 as a formula, and so, $\mathtt{f}_{sp}(\Pi_6, q)$ in Berthold et al.'s paper [17] does not satisfy strong persistence.

We characterise next a family of forks that are not representable as formulas, and include the case $(t \mid u \wedge s)$ in the previous example. We start by defining the following type of propositional formulas.

Definition 8 *(∅-contingent).* A propositional formula is said to be *∅-contingent* if both:

(i) $[\![\varphi]\!]^X \neq [\,]$ for some $X \subseteq At(\varphi)$;
(ii) and $\emptyset \notin [\![\varphi]\!]^X$, for every $X \subseteq At(\varphi)$.

It is easy to see that any ∅-contingent formula is also an HT-contingent formula, since it is not a contradiction, due to condition (i), whereas it is not a tautology either, due to condition (ii). The converse does not hold: as a counterexample, take $\varphi = \neg\neg p$. On the one hand, this formula is HT-contingent, since it is not a tautology because $\langle \emptyset, \emptyset \rangle \not\models \neg\neg p$, and it is not a contradiction because it has two HT-models, $\langle \{p\}, \{p\} \rangle$ and $\langle \emptyset, \{p\} \rangle$. On the other hand, it is not ∅-contingent since $\langle \emptyset, \{p\} \rangle \models \neg\neg p$, that is, $\emptyset \in [\![\neg\neg p]\!]^{\{p\}}$ and so, it violates condition (ii). Note that, in order to decide whether φ is ∅-contingent, it suffices to analyse the denotation of φ with respect to its local signature $At(\varphi)$. For instance, it is easy to see that any non-empty conjunction of atoms, such as t or $u \wedge s$ from the previous example, is ∅-contingent because it has models but no H-component in those models can be empty.

Proposition 7. *Given two ∅-contingent formulas φ and ψ with $At(\varphi) \cap At(\psi) = \emptyset$, the fork $(\varphi \mid \psi)$ is not reducible to a formula.*

Proposition 7 provides a sufficient condition to guarantee that a fork $(\varphi \mid \psi)$ cannot be reduced to a formula, but that condition is semantic: it requires examining the HT-models of φ and ψ (for their respective local signatures). Yet, some syntactic cases can be easily proven to fit into ∅-contingent formulas. For instance, any combination of atoms with non-empty conjunctions and disjunctions falls into this category, since no $H = \emptyset$ can ever form a model for these kinds of formulas.

Corollary 2. *Let φ and ψ be formulas exclusively formed with combinations of atoms, $\wedge$ and $\vee$ (no empty conjunction $\top$ or disjunction $\bot$ allowed) and let $At(\varphi) \cap At(\psi) = \emptyset$. Then, $(\varphi \mid \psi)$ is not reducible to a formula.*

Since the unfold operator is always applicable, our method can be used to forget multiple atoms by simply forgetting one by one in any arbitrary ordering. If the final result is reducible to a formula, the ordering in which we forget the atoms is irrelevant. Yet, it may be the case that, depending on that ordering, the intermediate results we obtain may be non-reducible to formulas, and must be kept as forks instead. To conclude this section, we illustrate multiple atoms forgetting with one more example.

Example 7. We want to forget both p and q in the following program Π_7:

$$\begin{array}{lll} \neg\neg p \wedge \neg\neg q \wedge a \to p & \neg p \to a & \neg p \wedge \neg q \to \bot \\ \neg\neg q \wedge p \to q & \neg q \to a & q \to a. \end{array} \qquad \square$$

As we show below, we cannot forget either p nor q alone and obtain a propositional formula, but when forgetting *the two of them*, the result can indeed be represented as a formula. Let us start forgetting p first and, on the result, forget q in a second step. To forget p, we can check that $\Pi_7[p/\bot]$ is

$$\begin{array}{lll} \neg\neg\bot \wedge \neg\neg q \wedge a \to \bot & \neg\bot \to a & \neg\bot \wedge \neg q \to \bot \\ \neg\neg q \wedge \bot \to q & \neg q \to a & q \to a \end{array}$$

whose conjunction amounts to $(\neg q \to \bot) \wedge a \wedge (\neg q \to a) \wedge (q \to a)$ where the last two conjuncts can be removed, leading to $(\neg q \to \bot) \wedge a$ or, if preferred, $\neg\neg q \wedge a$. On the other hand, $\Pi[\neg p/\bot]$ becomes

$$\begin{array}{lll} \neg\bot \wedge \neg\neg q \wedge a \to p & \bot \to a & \bot \wedge \neg q \to \bot \\ \neg\neg q \wedge p \to q & \neg q \to a & q \to a \end{array}$$

where we can remove the two rules with $\bot$ in the antecedent to obtain:

$$\neg\neg q \wedge a \to p \tag{33}$$

$$\neg\neg q \wedge p \to q \tag{34}$$

$$\neg q \to a \tag{35}$$

$$q \to a \tag{36}$$

Let $\Pi' = (33)$-(36). According to Theorem 4, we must now apply $\mathtt{f}_c(\Pi' \wedge \neg\neg p, p)$. To this aim, note that $behead^p(\Pi') = \Pi'$ because the only rule with p in the head is (33), and it is a supporting rule. Next, the only rule with p in the positive body is (34) and this must be replaced by $NES(\Pi' \wedge \neg\neg p, p, (34))$ that, in this case, corresponds to a single application of cut between (34) and (33) producing the rule $\neg\neg q \wedge \neg\neg q \wedge a \to \bot \vee q$ or simply $\neg\neg q \wedge a \to q$. Then rule (33), with p in the head, is removed. We also have to replace p in $\neg\neg p$ by $\neg NES(\Pi' \wedge \neg\neg p, p, \neg p)$, that is, by formula

$$\neg(\neg\neg q \wedge a \to \bot) \equiv \neg\neg q \wedge \neg\neg a.$$

To sum up, $\mathtt{f}_c(\Pi' \wedge \neg\neg p, p)$ amounts to:

$$\neg\neg q \wedge a \to q \qquad \neg q \to a \qquad q \to a \qquad \neg\neg q \qquad \neg\neg a$$

where, as $\neg\neg q$ holds, we can remove it in the antecedent of the first rule, whereas the second rule becomes trivially true, so we can further rewrite the program above as:

$$a \to q \qquad q \to a \qquad \neg\neg q \qquad \neg\neg a$$

or just $\neg\neg a \wedge (a \leftrightarrow q)$ because $\neg\neg q$ follows from the latter. Putting the two branches together, we have

$$\mathtt{f}_|(\Pi_7, p) \cong (a \wedge \neg\neg q) \mid (a \leftrightarrow q) \wedge \neg\neg a.$$

We can now forget q in this fork by forgetting it in both branches.

$$\begin{aligned} \mathtt{f}_|(\Pi_7, q) &\cong \mathtt{f}_|(a \wedge \neg\neg q, q) \mid \mathtt{f}_|((a \leftrightarrow q) \wedge \neg\neg a, q) \\ &\cong \mathtt{f}_c(a \wedge \neg\neg q, q) \mid \mathtt{f}_c((a \leftrightarrow q) \wedge \neg\neg a, q). \end{aligned} \tag{37}$$

For the left branch, note that $NES((a \wedge \neg\neg q), q, \neg q) = \top$ as there are no rules with q in the head and so we get an empty conjunction $\top$. After replacing q by $\neg NES((a \wedge \neg\neg q), q, \neg q) = \neg\top \cong \bot$ we obtain formula $\mathtt{f}_c(a \wedge \neg\neg q, q) = a \wedge \neg\neg\bot \cong \bot$. But then, by item (15) in Proposition 5, $(\bot \mid F) \cong F$ for any fork F and so, we can remove the whole left branch of (37) to obtain

$$\begin{aligned} \mathtt{f}_|(\Pi_7, q) &\cong \mathtt{f}_c((a \leftrightarrow q) \wedge \neg\neg a, q) \\ &\cong \mathtt{f}_c((a \to q) \wedge (q \to a) \wedge \neg\neg a, q) \\ &\cong (a \to a) \wedge \neg\neg a \\ &\cong \neg\neg a, \end{aligned}$$

since, as we can see, the only cut we can perform is between $(a \to q)$ and $(q \to a)$ leading to tautology $(a \to a)$.

To illustrate the effect of forgetting p and q in a different ordering, suppose now that we first forget q and then p in Π_7. On the one hand, we have that $\Pi_7[q/\bot]$ is:

$$\begin{array}{lll} \neg\neg p \wedge \neg\neg\bot \wedge a \to p & \neg p \to a & \neg p \wedge \neg\bot \to \bot \\ \neg\neg\bot \wedge p \to \bot & \neg\bot \to a & \bot \to a \end{array}$$

which can be simplified to $\neg\neg p \wedge (\neg p \to a) \wedge a$ or just $\neg\neg p \wedge a$. On the other hand, $\Pi_7[\neg q/\bot]$ is:

$$\begin{array}{lll} \neg\neg p \wedge \neg\bot \wedge a \to p & \neg p \to a & \neg p \wedge \bot \to \bot \\ \neg\bot \wedge p \to q & \bot \to a & q \to a \end{array}$$

or, equivalently:

$$\neg\neg p \wedge a \rightarrow p \tag{38}$$

$$\neg p \rightarrow a \tag{39}$$

$$p \rightarrow q \tag{40}$$

$$q \rightarrow a \tag{41}$$

If we name $\Pi'' =$ (38)-(41), we need to obtain the formula corresponding to $\mathtt{f}_c(\Pi'' \wedge \neg\neg q, q)$. The only rule with q in the positive body is (41) so it must be replaced by $NES(\Pi'', q, (41))$ which corresponds to $p \rightarrow a$. We must also replace q in $\neg\neg q$ by formula $\neg NES(\Pi'' \wedge \neg\neg q, q, \neg q) = \neg\neg p$. To sum up, $\mathtt{f}_c(\Pi'' \wedge \neg\neg q, q)$ becomes the formula

$$(p \rightarrow a) \wedge (\neg p \rightarrow a) \wedge (a \rightarrow p \vee \neg p) \wedge \neg\neg p$$

which can be simplified to $(a \leftrightarrow p) \wedge \neg\neg p$. So $\mathtt{f}_|(\Pi_7, q)$ is equivalent to

$$(a \wedge \neg\neg p \mid (a \leftrightarrow p) \wedge \neg\neg p).$$

Now, if we want to forget p in this fork, we proceed as before obtaining

$$\mathtt{f}_|(\mathtt{f}_|(\Pi_7, q), p) = \mathtt{f}_|(\mathtt{f}_|(\Pi_7, p), q) \equiv \neg\neg a.$$

Note how, even if we could not forget p nor q alone as a propositional formula, we can eventually forget the two atoms to achieve the formula $\neg\neg a$, regardless of the ordering in which we perform the forgetting of each individual atom.

4. Conclusions

We have presented a syntactic transformation called *unfolding* that is always applicable on any logic program and allows forgetting an atom (under strong persistence), producing an expression that may combine the fork operator and propositional formulas. Unfolding relies on another syntactic transformation, called the *cut* operator (close to $\mathtt{f}_{sp}$ by Berthold et al. [17]), that can be applied on any program that does not contain choice rules for the forgotten atom and, unlike unfolding, it returns a propositional formula without forks. Although in general the forks we obtain by unfolding cannot be reduced to propositional formulas, we have also illustrated how the use of general properties of forks makes this possible in some cases, even under circumstances where previous syntactic methods were not known to be applicable.

When compared to other syntactic methods [17,18] for multiple atoms, the definition of unfolding and cut is simpler in the sense that it performs common manipulations on formulas (the cut inference rule is well-known in sequent calculi) rather than trying to produce directly a logic program. Thus, if the forgetting results in a formula, we are free to leave it in that form or to further reduce it to a logic program using standard transformations in the logic of Here-and-There [19]. In some situations, abstaining from the reduction to a logic program, may be convenient. For instance, if we want to check whether some formula φ obtained after forgetting is strongly equivalent to another representation ψ, the direct use of formula φ is enough whereas the transformation into a logic program is unnecessary since it may produce, in the worst case, a combinatorial blow-up due to the application of distributivity rules.

Future work will be focused on extending the syntactic conditions under which forks can be reduced to formulas – we claim that this is an analogous situation to finding conditions under which second order quantifiers can be removed in second order logic. We will also study extensions of the unfolding operator like using sets of atoms, instead of proceeding one by one, or allowing arbitrary formulas rather than requiring a previous transformation to logic programs. Finally, another interesting topic to explore is the idea of succinctness, trying to find out whether there exist theoretical bounds for the variation in representational size when forgetting versus introducing an auxiliary atom.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Pedro Cabalar reports administrative support, article publishing charges, equipment, drugs, or supplies, and travel were provided by Spanish Ministry of Science and Innovation, MCIN/AEI/10.13039/501100011033 (grant PID2020-116201GB-I00). Felicidad Aguado reports administrative support, article publishing charges, equipment, drugs, or supplies, and travel were provided by Spanish Ministry of Science and Innovation, MCIN/AEI/10.13039/501100011033 (grant PID2020-116201GB-I00). Gilberto Perez reports administrative support, article publishing charges, equipment, drugs, or supplies, and travel were provided by Spanish Ministry of Science and Innovation, MCIN/AEI/10.13039/501100011033 (grant PID2020-116201GB-I00). Concepcion Vidal reports administrative support, article publishing charges, equipment, drugs, or supplies, and travel were provided by Spanish Ministry of Science and Innovation, MCIN/AEI/10.13039/501100011033 (grant PID2020-116201GB-I00). Pedro Cabalar reports administrative support, article publishing charges, equipment, drugs, or supplies, and travel were provided by Regional Government of Galicia, grant GPC ED431B 2022/33. Felicidad Aguado reports administrative support, article publishing charges, equipment, drugs, or supplies, and travel were provided by Regional Government of Galicia, grants CITIC (ED431G 2019/01), GPC ED431B 2022/33. Gilberto Perez reports administrative support, article publishing charges, equipment, drugs, or supplies, and travel were provided by Regional Government of Galicia, grants CITIC (ED431G 2019/01), GPC ED431B 2022/33. Concepcion Vidal reports administrative support,

article publishing charges, equipment, drugs, or supplies, and travel were provided by Regional Government of Galicia, grants CITIC (ED431G 2019/01), GPC ED431B 2022/33. Jorge Fandinno reports financial support, administrative support, and travel were provided by National Science Foundation. David Pearce reports administrative support, article publishing charges, equipment, drugs, or supplies, and travel were provided by Spanish Ministry of Science and Innovation, MCIN/AEI/10.13039/501100011033 (grant PID2020-116201GB-I00). Pedro Cabalar is Standard Editor of the Artificial Intelligence Journal.

Data availability

No data was used for the research described in the article.

Acknowledgements

We want to thank the anonymous reviewers for their suggestions that helped to improve this paper. Partially funded by Regional Government of Galicia and the European Union, grants CITIC (ED431G 2019/01), GPC ED431B 2022/33, by the Spanish Ministry of Science and Innovation, Spain, MCIN/AEI/10.13039/501100011033 (grant PID2020-116201GB-I00), by BBVA Foundation, Scientific Research Grants, (project LIANDA), and by the National Science Foundation (NSF 95-3101-0060-402).

Appendix A. Proofs of results

In the proofs, we will use the following notation. Given any rule r of the form $\varphi \to \psi$, we write $B(r)$ and $H(r)$ to stand for its body φ and head ψ, respectively. Moreover, given any atom $a \in \mathcal{AT}$ and any rule r, we define the formula

$$H^{\setminus a}(r) \stackrel{\text{def}}{=} \bigvee \{ p \mid p \in Hd(r), p \neq a \}$$

that corresponds to the head of r after (possibly) removing atom a. Similarly, $B^{\setminus a}(r)$ collects the conjunction of all body literals of r where atom a does not occur, that is, formally:

$$B^{\setminus a}(r) \stackrel{\text{def}}{=} \bigwedge \{ p \mid p \in Bd^{+}(r), p \neq a \} \ \wedge \ \bigwedge \{ \neg p \mid p \in Bd^{-}(r), p \neq a \} \\ \wedge \ \bigwedge \{ \neg\neg p \mid p \in Bd^{--}(r), p \neq a \}$$

The rule $B^{\setminus a}(r) \to H^{\setminus a}(r)$ will be denoted by $r^{\setminus a}$.

Proof of Proposition 3. If F, G are forks and $T \subseteq V \subseteq \mathcal{AT}$, we can deduce from the definition of $\langle\!\langle\, F \,\rangle\!\rangle_V^T$ that $\langle\!\langle\, F \mid G \,\rangle\!\rangle_V^T = \langle\!\langle\, F \,\rangle\!\rangle_V^T \cup \langle\!\langle\, G \,\rangle\!\rangle_V^T$. In consequence, when $F \cong_V F'$ and $G \cong_V G'$, we can use (i) from Theorem 1 in order to assert:

$$\langle\!\langle\, F \mid G \,\rangle\!\rangle_V^T = \langle\!\langle\, F \,\rangle\!\rangle_V^T \cup \langle\!\langle\, G \,\rangle\!\rangle_V^T = \langle\!\langle\, F' \,\rangle\!\rangle_V^T \cup \langle\!\langle\, G' \,\rangle\!\rangle_V^T = \langle\!\langle\, F' \mid G' \,\rangle\!\rangle_V^T \quad \square$$

Proof of Proposition 5. Notice that $\langle\!\langle\, \bot \,\rangle\!\rangle^Y = \emptyset$ and $\langle\!\langle\, \top \,\rangle\!\rangle^Y = \{2^Y\}$ for any $Y \subseteq \mathcal{AT}$. On the other hand, $\langle\!\langle\, \neg\alpha \,\rangle\!\rangle^Y = \emptyset$ and $\langle\!\langle\, \neg\neg\alpha \,\rangle\!\rangle^Y = \{2^Y\}$ if $Y \vDash \alpha$ and $\langle\!\langle\, \neg\neg\alpha \,\rangle\!\rangle^Y = \emptyset$ and $\langle\!\langle\, \neg\alpha \,\rangle\!\rangle^Y = \{2^Y\}$ if $Y \nvDash \alpha$.
We also have:

$$\langle\!\langle\, \alpha \mid \neg\alpha \,\rangle\!\rangle^Y = \langle\!\langle\, \alpha \,\rangle\!\rangle^Y \cup \langle\!\langle\, \neg\alpha \,\rangle\!\rangle^Y = \begin{cases} \langle\!\langle\, \alpha \,\rangle\!\rangle^Y & \text{if } Y \vDash \alpha \\ \{2^Y\} & \text{if } Y \nvDash \alpha \end{cases}$$

and

$$[\![\alpha \vee \neg\alpha]\!]^Y = \begin{cases} [\![\alpha]\!]^Y & \text{if } Y \vDash \alpha \\ 2^Y & \text{if } Y \nvDash \alpha \end{cases}$$

In general, if F, G and H are forks, then we can apply Proposition 12 from [16]:

$$F \wedge (G \mid H) \equiv (F \wedge G \mid F \wedge H)$$

Consequently:

$$\alpha \equiv \alpha \wedge \top \equiv \alpha \wedge (\neg\beta \mid \neg\neg\beta) \equiv (\alpha \wedge \neg\beta \mid \alpha \wedge \neg\neg\beta) \quad \square$$

For any $Y \subseteq \mathcal{AT}$ and $a \in \mathcal{AT}$, we denote by $Y(a) = Y \cup \{a\}$.

Lemma 2. *Let Π be a logic program for signature $\mathcal{AT}$ not containing a-choices for some atom $a \in \mathcal{AT}$. For any $Y \subseteq V = \mathcal{AT} \setminus \{a\}$, if $Y \vDash \Pi$ and $Y(a) \vDash \Pi$, then $\langle Y, Y(a) \rangle \vDash \Pi$.*

Proof. For the proof, we only have to notice that:

1. $\langle Y, Y(a)\rangle \models r$ for any r such that $a \in B^{+}(r) \cup B^{-}(r)$ or $a \notin \mathcal{AT}(r)$
2. If $a \in B^{--}(r)$, then:

$$\langle Y, Y(a)\rangle \not\models B(r) \iff Y(a) \not\models B(r), \text{ and } \langle Y, Y(a)\rangle \models H(r) \iff Y(a) \models H(r)$$

3. If $a \in H(r)$, then:

$$\langle Y, Y(a)\rangle \not\models B(r) \iff Y \not\models B(r), \text{ and } \langle Y, Y(a)\rangle \models H(r) \iff Y \models H(r) \quad \square$$

For the proof of Theorem 2 we split the result in the next three Lemmata, one for each of the three cases of a-forgettable programs.

Lemma 3. *Let Π be a logic program for signature $\mathcal{AT}$, let $V \subseteq \mathcal{AT}$ and $a \in \mathcal{AT} \setminus V$. If Π contains the rule $\top \to a$, then: $\Pi \cong_V \mathtt{f}_c(\Pi, a)$.* $\square$

Proof. We must prove that, if $Y \subseteq V = \mathcal{AT} \setminus \{a\}$, then $\langle\!\langle\, \Pi \,\rangle\!\rangle_V^Y = \langle\!\langle\, \mathtt{f}_c(\Pi, a) \,\rangle\!\rangle^Y$. In this case, $NES(\Pi, a, \neg a) \cong \bot$, so after the application of the operator $\mathtt{f}_c$ to $\Pi' = behead^a(\Pi)$, we will have that $\mathtt{f}_c(\Pi, a) \equiv \{r^{\setminus a} \mid r \in \Pi\} \equiv \Pi[a/\top]$.
Notice that, when $r, r' \in \Pi$ satisfy that $a \in B^{+}(r)$ and $a \in H(r')$ but $a \notin B^{--}(r')$, then:

$$Cut(a, r, r') \wedge \{r^{\setminus a}, r'^{\setminus a}\} \equiv \{r^{\setminus a}, r'^{\setminus a}\}$$

and

$$Cut(a, r, \top \to a) = r^{\setminus a}.$$

Finally, we can apply (1) to conclude:

$$\Pi \equiv \Pi \wedge a \;\equiv\; \Pi[a/\top] \wedge a \equiv_V \Pi[a/\top]$$

which finishes the proof. $\square$

Lemma 4. *Let Π be a logic program for signature $\mathcal{AT}$, let $V \subseteq \mathcal{AT}$ and $a \in \mathcal{AT} \setminus V$. If all rules in Π in which a occurs are a-choices, then: $\Pi \cong_V \mathtt{f}_c(\Pi, a)$.* $\square$

Proof. As in the previous lemma, we must prove that, if $Y \subseteq V = \mathcal{AT} \setminus \{a\}$, then $\langle\!\langle\, \Pi \,\rangle\!\rangle_V^Y = \langle\!\langle\, \mathtt{f}_c(\Pi, a) \,\rangle\!\rangle^Y$. After applying the operator $\mathtt{f}_c$ to $\Pi' = behead^a(\Pi)$, any a-choice r will be transformed into a tautology since both $\neg\neg a$ in $B(r)$ and a in $H(r)$ will be substituted by $\neg NES(\Pi, a, \neg a)$. This means $\mathtt{f}_c(\Pi, a) \equiv R'$ where R' contains all the rules $r \in \Pi$ such that $a \notin \mathcal{AT}(r)$.
When $Y \subseteq V$, then $[\![\, \Pi \,]\!]^Y = [\![\, R' \,]\!]^Y$ because $Y \not\models B(r)$ for any a-choice rule r.
Moreover: $\langle\!\langle\, \Pi \,\rangle\!\rangle_V^Y = \langle\!\langle\, \Pi \,\rangle\!\rangle^Y$ because $[\![\, \Pi \,]\!]_V^{Y(a)} = [\![\, \Pi \,]\!]^Y$. Notice that, for any $X \subseteq Y$, we have that $\langle X, Y\rangle$ and $\langle X(a), Y(a)\rangle$ satisfy any a-choice ($Y \not\models \neg\neg a$ and $\langle X(a), Y(a)\rangle \models a$). It also holds that, for any $r' \in R'$, we get:

$$\langle X, Y\rangle \models r' \iff \langle X(a), Y(a)\rangle \models r'$$

We can conclude that $[\![\, \Pi \,]\!]_V^{Y(a)} = [\![\, \Pi \,]\!]^Y$ and, in consequence:

$$\langle\!\langle\, \Pi \,\rangle\!\rangle_V^Y = \langle\!\langle\, \Pi \,\rangle\!\rangle^Y = \langle\!\langle\, R' \,\rangle\!\rangle^Y$$

which implies that $\Pi \equiv_V R' \equiv \mathtt{f}_c(\Pi, a)$. $\square$

Lemma 5. *Let Π be a logic program for signature $\mathcal{AT}$, let $V \subseteq \mathcal{AT}$ and $a \in \mathcal{AT} \setminus V$. If Π does not contain a-choices, then: $\Pi \cong_V \mathtt{f}_c(\Pi, a)$.* $\square$

Proof. As before, we want to prove that $Y \subseteq V = \mathcal{AT} \setminus \{a\}$ implies $\langle\!\langle\, \Pi \,\rangle\!\rangle_V^Y = \langle\!\langle\, \mathtt{f}_c(\Pi, a) \,\rangle\!\rangle^Y$. Throughout all the proof, we use that, for any $X \subseteq Y \subseteq V$, one of the following conditions is satisfied.

- $\langle X, Y\rangle \models \neg\neg NES(\Pi, a, \neg a)$ iff $Y \models \neg\neg NES(\Pi, a, \neg a)$ iff $Y \models r$, for all $r \in \Pi$ such that $a \in H(r)$.
- $\langle X, Y\rangle \models \neg NES(\Pi, a, \neg a)$ iff $Y \models \neg NES(\Pi, a, \neg a)$ iff $Y \not\models r$, for some $r \in \Pi$ such that $a \in H(r)$.

We distinguish all possible cases taking into account that, whenever $\langle Y, Y(a)\rangle \models \Pi$, then $Y(a) \models \Pi$ (by Persistence).

1. $\boxed{Y \not\models \Pi \text{ and } Y(a) \not\models \Pi.}$
   In this case, $\langle\!\langle\, \Pi \,\rangle\!\rangle_V^Y = \emptyset$. We will show that $Y \not\models \mathtt{f}_c(\Pi, a)$. First of all, when $Y \not\models \Pi$, then we can have two different situations:
   - $Y \models r$, for any $r \in \Pi$ such that $a \in H(r)$ but there exists $r_1 \in \Pi$ such that $a \in B^{-}(r_1)$, $Y \models B(r_1)$ but $Y \not\models H(r_1)$. In this case, $Y \models \neg\neg NES(\Pi, a, \neg a) \wedge B^{\setminus a}(r_1)$ but $Y \not\models H(r_1)$ and, in consequence $Y \not\models \mathtt{f}_c(\Pi, a)$.
   - There exists $r_4 \in \Pi$ such that $a \in H(r_4)$, $Y \models B(r_4)$ but $Y \not\models H(r_4)$ (or $Y \not\models H^{\setminus a}(r_4)$ since $a \notin Y$). Since $Y(a) \not\models \Pi$, two different scenarios can be possible:

– There exists $r_0 \in \Pi$ such that $a \in B^+(r_0)$, $Y(a) \models B(r_0)$ but $Y(a) \not\models H(r_0)$. Then $Y \models B^{\setminus a}(r_0)$ and $Y \not\models H(r_0)$ (notice that $a \notin H(r_0)$). The rule $Cut(a, r_0, r_4)$:

$$B^{\setminus a}(r_0) \wedge B(r_4) \rightarrow H(r_0) \vee H^{\setminus a}(r_4)$$

is not satisfied by Y and we have finished.

– There exists $r_2 \in \Pi$ such that $a \in B^{--}(r_2)$, $Y(a) \models B(r_2)$ but $Y(a) \not\models H(r_2)$ (this is equivalent to say that $Y \models B^{\setminus a}(r_2)$ and $Y \not\models H(r_2)$). Then $Y \models B^{\setminus a}(r_2)$ and $Y \models \neg NES(\Pi, a, \neg a)$ (notice that $Y \not\models r_4$). The rule:

$$\neg NES(\Pi, a, \neg a) \wedge B^{\setminus a}(r_2) \rightarrow H(r_2)$$

is not satisfied by Y.

+

2. $Y \not\models \Pi$ and $\langle Y, Y(a)\rangle \models \Pi$ (which implies $Y(a) \models \Pi$).

From these hypothesis, we conclude $\langle\!\langle \Pi \rangle\!\rangle_V^Y = \emptyset$, since $[\![\Pi]\!]^{Y(a)}$ is not V-feasible. We should prove that $Y \not\models \mathtt{f}_c(\Pi, a)$.
First of all, notice that $Y \not\models \Pi$ but $\langle Y, Y(a)\rangle \models \Pi$ which implies $Y \models r$, for any $r \in \Pi$ with $a \in H(r)$ and $Y \not\models r_1$, for some $r_1 \in \Pi$ with $a \in B^-(r_1)$. We can conclude:

$$Y \not\models \neg\neg NES(\Pi, a, \neg a) \wedge B^{\setminus a}(r_1) \rightarrow H(r_1)$$

because $Y \models \neg\neg NES(\Pi, a, \neg a) \wedge B^{\setminus a}(r_1)$ and $Y \not\models H(r_1)$.

3. $Y \models \Pi$ and $\langle Y, Y(a)\rangle \models \Pi$ (which implies that $Y(a) \models \Pi$).

In this case, it holds $Y \models \neg\neg NES(\Pi, a, \neg a)$ and $\langle\!\langle \Pi \rangle\!\rangle_V^Y = \downarrow [\![\Pi]\!]^Y$. Let's prove that:

$$[\![\mathtt{f}_c(\Pi, a)]\!]^Y = [\![\Pi]\!]^Y$$

"$\subseteq$"
Suppose that $\langle X, Y\rangle \models \mathtt{f}_c(\Pi, a)$. We only have to show that $\langle X, Y\rangle \models r$ if $a \in B^-(r)$ or $a \in H(r)$.
• Take r such that $a \in B^-(r)$ and suppose that $\langle X, Y\rangle \models B(r)$. Then $\langle X, Y\rangle \models \neg\neg NES(\Pi, a, \neg a) \wedge B^{\setminus a}(r)$ so $\langle X, Y\rangle \models H(r)$.
• Now suppose that $\langle X, Y\rangle \models B(r)$ for some r with $a \in H(r)$. Then $\langle X, Y\rangle \models H^{\setminus a}(r)$ because $\langle X, Y\rangle$ satisfies the rule:

$$B(r) \rightarrow \neg NES(\Pi, a, \neg a) \vee H^{\setminus a}(r)$$

and $\langle X, Y\rangle \not\models \neg NES(\Pi, a, \neg a)$.
"$\supseteq$"
Suppose that $\langle X, Y\rangle \models \Pi$. Since $\langle X, Y\rangle \not\models \neg NES(\Pi, a, \neg a)$, we have that $\langle X, Y\rangle$ satisfies

$$B^{\setminus a}(r_2) \wedge \neg NES(\Pi, a, \neg a) \rightarrow H^{\setminus a}(r_2)$$

if $r_2 \in \Pi$ and $a \in B^{--}(r_2)$. For the other rules in $\mathtt{f}_c(\Pi, a)$, we have:
• If $r_1 \in \Pi$, $a \in B^-(r_1)$ and $\langle X, Y\rangle \models \neg\neg NES(\Pi, a, \neg a) \wedge B^{\setminus a}(r_1)$, then $\langle X, Y\rangle \models B(r_1)$ (since $B(r_1) = \neg a \wedge B^{\setminus a}(r_1)$), so $\langle X, Y\rangle \models H(r_1)$.
• If $r_4 \in \Pi$ and $a \in H(r_4)$ and $\langle X, Y\rangle \models B(r_4)$, then $\langle X, Y\rangle$ satisfies the rule:

$$B(r_4) \rightarrow \neg NES(\Pi, a, \neg a) \vee H^{\setminus a}(r_4)$$

because $\langle X, Y\rangle \models H(r_4)$ which implies $\langle X, Y\rangle \models H^{\setminus a}(r_4)$.
• Take $r_0, r_4 \in \Pi$ with $a \in B^+(r_0)$ and $a \in H(r_4)$. Suppose that $\langle X, Y\rangle \models B^{\setminus a}(r_0) \wedge B(r_4)$. Since $\langle X, Y\rangle \models r_4$ and $a \notin X$, we conclude $\langle X, Y\rangle \models H^{\setminus a}(r_4)$ and so $\langle X, Y\rangle \models Cut(a, r_0, r_4)$.

4. $Y \models \Pi$, $Y(a) \models \Pi$ and $\langle Y, Y(a)\rangle \not\models \Pi$.

This case is not possible when Π does not have a-choices as we have seen in Lemma 2.

5. $Y \models \Pi$ and $Y(a) \not\models \Pi$.

Now, $\langle\!\langle \Pi \rangle\!\rangle_V^Y = \downarrow [\![\Pi]\!]^Y$. Let's prove that:

$$[\![\mathtt{f}_c(\Pi, a)]\!]^Y = [\![\Pi]\!]^Y$$

$Y \models \Pi$ so $\langle X, Y\rangle \models \neg\neg NES(\Pi, a, \neg a)$ for any $X \subseteq Y$.
"$\subseteq$"
Suppose that $\langle X, Y\rangle \models \mathtt{f}_c(\Pi, a)$. We only have to prove that $\langle X, Y\rangle \models r$ when $a \in B^-(r)$ or $a \in H(r)$.
• First of all, suppose that $\langle X, Y\rangle \models B(r_1)$ with $r_1 \in \Pi$ and $a \in B^-(r_1)$. Then $\langle X, Y\rangle \models \neg\neg NES(\Pi, a, \neg a) \wedge B^{\setminus a}(r_1)$ which implies $\langle X, Y\rangle \models H(r_1)$.
• Now suppose we have $r_4 \in \Pi$ such that $a \in H(r_4)$ and $\langle X, Y\rangle \models B(r_4)$, then $\langle X, Y\rangle \models \neg NES(\Pi, a, \neg a) \vee H^{\setminus a}(r_4)$ which implies $\langle X, Y\rangle \models H(r_4)$ since $\langle X, Y\rangle \not\models \neg NES(\Pi, a, \neg a)$.
"$\supseteq$"
Suppose that $\langle X, Y\rangle \models \Pi$. Since $\langle X, Y\rangle \not\models \neg NES(\Pi, a, \neg a)$, then $\langle X, Y\rangle$ satisfies:

$$\neg NES(\Pi, a, \neg a) \wedge B^{\setminus a}(r_2) \rightarrow H(r_2)$$

when $a \in B^{--}(r_2)$ and $r_2 \in \Pi$. For the other rules:

- Take $r_1 \in \Pi$ with $a \in B^{-}(r_1)$ and suppose that $\langle X, Y\rangle \models \neg\neg NES(\Pi, a, \neg a) \wedge B^{\setminus a}(r_1)$, then $\langle X, Y\rangle \models B(r_1) = (\neg a \wedge B^{\setminus a}(r_1))$ which implies $\langle X, Y\rangle \models H(r_1)$.
- Take $r_4 \in \Pi$ with $a \in H(r_4)$ and suppose that $\langle X, Y\rangle \models B(r_4)$, then $\langle X, Y\rangle \models H(r_4)$ which implies $\langle X, Y\rangle \models H^{\setminus a}(r_4)$. This proves $\langle X, Y\rangle \models Cut(a, r_0, r_4)$ for any $r_0 \in \Pi$ such that $a \in B^{+}(r_0)$ and also that $\langle X, Y\rangle$ is a model of:

$$B(r_4) \rightarrow \neg NES(\Pi, a, \neg a) \vee H^{\setminus a}(r_4).$$

6. $\boxed{Y \not\models \Pi,\ Y(a) \models \Pi \text{ and } \langle Y, Y(a)\rangle \not\models \Pi.}$

In this case, $\langle\!\langle\, \Pi \,\rangle\!\rangle_V^Y = \downarrow [\![\, \Pi \,]\!]_V^{Y(a)}$. Let's prove that:

$$[\![\, \mathtt{f}_c(\Pi, a) \,]\!]^Y = [\![\, \Pi \,]\!]_V^{Y(a)}$$

The fact that $\langle Y, Y(a)\rangle \not\models \Pi$ implies $\langle Y, Y(a)\rangle \not\models r$ for some $r \in \Pi$ satisfying $a \in B^{--}(r)$ or $a \in H(r)$. But, if $a \in B^{--}(r)$ and $\langle Y, Y(a)\rangle \models B(r)$, we also have $Y(a) \models B(r)$ which implies $Y(a) \models H(r)$ or, equivalently, $\langle Y, Y(a)\rangle \models H(r)$ (notice that $a \notin H(r)$).
So, in this case, there exists $r \in \Pi$ with $a \in H(r)$ such that $\langle Y, Y(a)\rangle \not\models r$ which also implies $Y \not\models r$, because, for this rule r, we have $\langle Y, Y(a)\rangle \models B(r)$ iff $Y \models B(r)$ ($a \notin B(r)$) and $\langle Y, Y(a)\rangle \models H(r)$ iff $Y \models H(r)$.
We can use that $\langle X, Y\rangle \models \neg NES(\Pi, a, \neg a)$ for any $X \subseteq Y$.
"$\subseteq$"
Suppose that $\langle X, Y\rangle \models \mathtt{f}_c(\Pi, a)$. First of all, notice that, for any $r \in \Pi$ such that $a \notin \mathcal{AT}(r)$, it holds:

$$\langle X, Y\rangle \models r \text{ iff } \langle X, Y(a)\rangle \models r \text{ iff } \langle X(a), Y(a)\rangle \models r$$

If $X \notin [\![\, \Pi \,]\!]_V^{Y(a)}$, then: $\langle X, Y(a)\rangle \not\models \Pi$ and $\langle X(a), Y(a)\rangle \not\models \Pi$.
This implies that $\langle X, Y(a)\rangle \not\models r$ for some rule r satisfying $a \in B^{--}(r)$ or $a \in H(r)$ and $\langle X(a), Y(a)\rangle \not\models r'$ for some rule r' satisfying $a \in B^{+}(r')$ or $a \in B^{--}(r')$.

- First of all, suppose that $\langle X, Y(a)\rangle \not\models r$ with $a \in B^{--}(r)$ which means $\langle X, Y(a)\rangle \models B(r)$ and $\langle X, Y(a)\rangle \not\models H(r)$. But then $\langle X, Y\rangle \models \neg NES(\Pi, a, \neg a) \wedge B^{\setminus a}(r)$ and $\langle X, Y(a)\rangle \not\models H(r)$ or $\langle X, Y\rangle \not\models H(r)$ which is a contradiction.
- Now suppose that we have $r, r' \in \Pi$ with $a \in B^{+}(r) \cap H(r')$ and such that $\langle X, Y(a)\rangle \models B(r')$, $\langle X, Y(a)\rangle \not\models H(r')$, $\langle X(a), Y(a)\rangle \models B(r)$ and $\langle X(a), Y(a)\rangle \not\models H(r)$. Then $\langle X, Y\rangle \models B^{\setminus a}(r) \wedge B(r')$ but $\langle X, Y\rangle \not\models H(r) \vee H^{\setminus a}(r')$. This implies $\langle X, Y\rangle \not\models Cut(a, r, r')$.
- We only have to show that, if $\langle X(a), Y(a)\rangle \not\models r$ with $a \in B^{--}(r)$, then $\langle X, Y\rangle \not\models \mathtt{f}_c(\Pi, a)$. Since $\langle X(a), Y(a)\rangle \models B(r)$ and $\langle X(a), Y(a)\rangle \not\models H(r)$, then $\langle X, Y\rangle \models \neg NES(\Pi, a, \neg a) \wedge B^{\setminus a}(r)$ but $\langle X, Y\rangle \not\models H(r)$ which implies $\langle X, Y\rangle \not\models \mathtt{f}_c(\Pi, a)$.

"$\supseteq$"
Take $X \subseteq Y(a)$ such that $\langle X, Y(a)\rangle \models \Pi$. We are going to show that $\langle X \cap V, Y\rangle \models \mathtt{f}_c(\Pi, a)$. Firs of all, notice that $\langle X \cap V, Y\rangle \models \neg NES(\Pi, a, \neg a)$. This implies $\langle X \cap V, Y\rangle$ is a model of:

$$\neg\neg NES(\Pi, a, \neg a) \wedge B^{\setminus a}(r) \rightarrow H(r)$$

when $a \in B^{-}(r)$ and also of:

$$B(r) \rightarrow \neg NES(\Pi, a, \neg a) \vee H^{\setminus a}(r)$$

when $a \in H(r)$.
For the other rules:

- Take $r \in \Pi$ with $a \in B^{--}(r)$ and suppose that $\langle X \cap V, Y\rangle \models \neg NES(\Pi, a, \neg a) \wedge B^{\setminus a}(r)$. Then $\langle X, Y(a)\rangle \models \neg\neg a \wedge B^{\setminus a}(r)$ since $a \notin B^{\setminus a}(r)$. In consequence, $\langle X, Y(a)\rangle \models H(r)$ or $\langle X \cap V, Y\rangle \models H(r)$ because $a \notin H(r)$.
- We are going to show that $\langle X \cap V, Y\rangle \models Cut(a, r, r')$ if $r, r' \in \Pi$, $a \in B^{+}(r) \cap H(r')$. Suppose that $\langle X \cap V, Y\rangle \models B^{\setminus a}(r) \wedge B(r')$. We can distinguish two cases:
  - If $a \in X$, then $\langle X, Y(a)\rangle \models B(r)$ and $\langle X, Y(a)\rangle \models H(r)$ or $\langle X \cap V, Y\rangle \models H(r)$.
  - If $a \notin X$, then $X = X \cap V$ and$\langle X, Y(a)\rangle \models B(r')$. In consequence $\langle X, Y(a)\rangle \models H(r')$ or $\langle X \cap V, Y\rangle \models H^{\setminus a}(r')$. □

Proof of Theorem 2. It follows from Lemmata 3, 4 and 5 that respectively cover the three cases in Definition 6 of a-forgettable program. □

Proof of Theorem 3. Notice that $\Pi \cong (\Pi \wedge \neg a \mid \Pi \wedge \neg\neg a)$. Since $\Pi \wedge \neg a$ and $\Pi \wedge \neg\neg a$ are a-forgettable because they do not have a-choices (an a-choice in Π is a tautology in $\Pi \wedge \neg a$ and a rule like $B \rightarrow a \vee H$ with $a \notin B \cup H$ in $\Pi \wedge \neg\neg a$), we can deduce the result by using Theorem 2 and Proposition 3. □

Proof of Lemma 1. Just take into account that $\Pi \equiv_V \mathtt{f}_{|}(\Pi, a)$, where $V = \mathcal{AT} \setminus \{a\}$ and the fact that $\mathcal{AT}(\Pi') \subseteq V$. □

Proof of Proposition 7. We will prove that there exists some T for which $\langle\!\langle\, \varphi \mid \psi \,\rangle\!\rangle^T$ has more than one $\preceq$-maximal support, which means that $(\varphi \mid \psi)$ is not reducible to any formula, according to Proposition 2. Note first that:

$$\langle\!\langle\, \varphi \mid \psi \,\rangle\!\rangle^T = \langle\!\langle\, \varphi \,\rangle\!\rangle^T \cup \langle\!\langle\, \psi \,\rangle\!\rangle^T = {\downarrow}[\![\varphi]\!]^T \cup {\downarrow}[\![\psi]\!]^T$$

So, it is enough to find some T such that $[\![\varphi]\!]^T \not\subseteq [\![\psi]\!]^T$ and $[\![\psi]\!]^T \not\subseteq [\![\varphi]\!]^T$. Due to condition (i) in Definition 8, there exist $T_1 \subseteq At(\varphi)$ and $T_2 \subseteq At(\psi)$ such that $T_1 \models \varphi$ and $T_2 \models \psi$. Take $T = T_1 \cup T_2$: without loss of generality, we will just prove that $[\![\varphi]\!]^T \not\subseteq [\![\psi]\!]^T$ and the other direction $[\![\psi]\!]^T \not\subseteq [\![\varphi]\!]^T$ is analogous. Since $[\![\varphi]\!]^T \neq [\,]$ and $[\![\psi]\!]^T \neq [\,]$, we can show that $[\![\psi]\!]^T \not\subseteq [\![\varphi]\!]^T$. Finally, notice that $T_2 \in [\![\psi]\!]^T$ but $T_2 \notin [\![\varphi]\!]^T$, since, otherwise, we would have that $\emptyset = T_2 \cap At(\varphi) \in [\![\varphi]\!]^{T_1}$ which is not possible due to condition (ii) from Definition 8. □

Proof of Corollary 1. By using Theorem 2 and Theorem 3, we know that $\mathtt{f}_c(\Pi, a) \cong_V \Pi \cong_V \mathtt{f}_{|}(\Pi, a)$, and so, $\mathtt{f}_{|}(\Pi, a) \cong \mathtt{f}_c(\Pi, a)$. □

Proof of Theorem 4. The replacement of the right branch $\mathtt{f}_c(\Pi \wedge \neg\neg a, a)$ by $\mathtt{f}_c(\Pi[\neg a/\bot] \wedge \neg\neg a, a)$ follows directly from (3). Similarly, the replacement of the left branch $\mathtt{f}_c(\Pi \wedge \neg a, a)$ by $\mathtt{f}_c(\Pi[a/\bot] \wedge \neg a, a)$ follows from (2). For the second equivalence, note that a does not occur in $\Pi[a/\bot]$ whereas rule $\neg a = (a \rightarrow \bot)$ does not contain an a-choice. This means that the $\mathtt{f}_c$ operator is applicable and, in this case, it just removes $\neg a$ leaving $\Pi[a/\bot]$. □

References

[1] V. Marek, M. Truszczyński, Stable models and an alternative logic programming paradigm, in: The Logic Programming Paradigm: a 25-Year Perspective, Springer-Verlag, 1999, pp. 169–181.
[2] I. Niemelä, Logic programs with stable model semantics as a constraint programming paradigm, Ann. Math. Artif. Intell. 25 (1999) 241–273.
[3] F. Aguado, P. Cabalar, M. Diéguez, G. Pérez, T. Schaub, A. Schuhmann, C. Vidal, Linear-time temporal answer set programming, Theory Pract. Log. Program. (2021) 1–55.
[4] S. Tran Cao, E. Pontelli, M. Balduccini, T. Schaub, Answer set planning: a survey, Theory Pract. Log. Program. (2022) 1–73.
[5] J. Fandinno, W. Faber, M. Gelfond, Thirty years of epistemic specifications, Theory Pract. Log. Program. 22 (6) (2022) 1043–1083.
[6] Y. Lierler, Constraint answer set programming: integrational and translational (or SMT-based) approaches, Theory Pract. Log. Program. (2021) 1–31.
[7] J. Leite, M. Slota, A brief history of updates of answer-set programs, Theory Pract. Log. Program. (2022) 1–51.
[8] E. Erdem, M. Gelfond, N. Leone, Applications of answer set programming, AI Mag. 37 (3) (2016) 53–68.
[9] R. Gonçalves, M. Knorr, J. Leite, Forgetting in answer set programming - a survey, Theory Pract. Log. Program. 23 (1) (2023) 111–156.
[10] M. Knorr, J.J. Alferes, Preserving strong equivalence while forgetting, in: E. Fermé, J. Leite (Eds.), Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings, in: Lecture Notes in Computer Science, vol. 8761, Springer, 2014, pp. 412–425.
[11] R. Gonçalves, M. Knorr, J. Leite, You can't always forget what you want: on the limits of forgetting in answer set programming, in: G.A. Kaminka, M. Fox, P. Bouquet, E. Hüllermeier, V. Dignum, F. Dignum, F. van Harmelen (Eds.), Proceedings of 22nd European Conference on Artificial Intelligence (ECAI'16), in: Frontiers in Artificial Intelligence and Applications, vol. 285, IOS Press, 2016, pp. 957–965.
[12] A. Heyting, Die formalen Regeln der intuitionistischen Logik, in: Sitzungsberichte der Preussischen, Akademie der Wissenschaften, Deutsche Akademie der Wissenschaften zu Berlin, 1930, pp. 42–56, reprint in Logik-Texte: Kommentierte Auswahl zur Geschichte der Modernen Logik, Akademie-Verlag, 1986.
[13] D. Pearce, A new logical characterisation of stable models and answer sets, in: J. Dix, L.M. Pereira, T.C. Przymusinski (Eds.), Selected Papers from the Non-Monotonic Extensions of Logic Programming (NMELP'96), in: Lecture Notes in Artificial Intelligence, vol. 1216, Springer-Verlag, 1996, pp. 57–70.
[14] P. Cabalar, P. Ferraris, Propositional theories are strongly equivalent to logic programs, Theory Pract. Log. Program. 7 (6) (2007) 745–759.
[15] P. Cabalar, D. Pearce, A. Valverde, Minimal logic programs, in: V. Dahl, I. Niemelä (Eds.), Proceedings of the 23rd International Conference on Logic Programming, (ICLP'07), Springer, 2007, pp. 104–118.
[16] F. Aguado, P. Cabalar, J. Fandinno, D. Pearce, G. Pérez, C. Vidal, Forgetting auxiliary atoms in forks, Artif. Intell. 275 (2019) 575–601.
[17] M. Berthold, R. Gonçalves, M. Knorr, J. Leite, A syntactic operator for forgetting that satisfies strong persistence, Theory Pract. Log. Program. 19 (5–6) (2019) 1038–1055.
[18] M. Berthold, On syntactic forgetting with strong persistence, in: Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning, 2022, pp. 43–52.
[19] P. Cabalar, D. Pearce, A. Valverde, Reducing propositional theories in equilibrium logic to logic programs, in: C. Bento, A. Cardoso, G. Dias (Eds.), Proceedings of the 12th Portuguese Conference on Progress in Artificial Intelligence (EPIA'05), in: Lecture Notes in Computer Science, vol. 3808, Springer, 2005, pp. 4–17.
[20] F. Aguado, P. Cabalar, J. Fandinno, D. Pearce, G. Pérez, C. Vidal, Syntactic asp forgetting with forks, in: G. Gottlob, D. Inclezan, M. Maratea (Eds.), Logic Programming and Nonmonotonic Reasoning, LPNMR 2022, in: Lecture Notes in Computer Science, vol. 13416, Springer, 2022, pp. 3–15.
[21] F. Aguado, P. Cabalar, D. Pearce, G. Pérez, C. Vidal, A denotational semantics for equilibrium logic, Theory Pract. Log. Program. 15 (4–5) (2015) 620–634.
[22] P. Ferraris, J. Lee, V. Lifschitz, A generalization of the Lin-Zhao theorem, Ann. Math. Artif. Intell. 47 (1–2) (2006) 79–101.
[23] S. Brass, J. Dix, Semantics of (disjunctive) logic programs based on partial evaluation, J. Log. Program. 40 (1) (1999) 1–46.
[24] F. Aguado, P. Cabalar, J. Fandinno, G. Pérez, C. Vidal, A logic program transformation for strongly persistent forgetting – extended abstract, in: Proc. of the 37th Intl. Conf. on Logic Programming (ICLP'21), Porto, Portugal (virtual event), in: Electronic Proceedings in Theoretical Computer Science (EPTCS), vol. 345, 2021, pp. 11–13.
[25] G. Mints, Cut-free formulations for a quantified logic of here and there, Ann. Pure Appl. Log. 162 (3) (2010) 237–242.