TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute of Artificial Intelligence Knowledge Representation and Reasoning

# Forward Human Reasoning Modeled by Logic Programming Modeled by Classical Logic with Circumscription and Projection

Christoph Wernhard

KRR Report 11-07

Mail to
Technische Universität Dresden
01062 Dresden

Bulk mail to
Technische Universität Dresden
Helmholtzstr. 10
01069 Dresden

Office
Room 2006
Nöthnitzer Straße 46
01187 Dresden

Internet
http://www.wv.inf.tu-dresden.de

# Forward Human Reasoning Modeled by Logic Programming Modeled by Classical Logic with Circumscription and Projection

**– 14 August 2011 –**

Christoph Wernhard

Technische Universität Dresden
christoph.wernhard@tu-dresden.de

**Abstract.** Recently an approach to model human reasoning as studied in cognitive science by logic programming, has been introduced by Stenning and van Lambalgen and exemplified with the suppression task. We investigate this approach from the view of a framework where different logic programming semantics correspond to different translations of logic programs into formulas of classical two-valued logic extended by two second-order operators, circumscription and literal projection. Based on combining and extending previously known such renderings of logic programming semantics, we take semantics into account that have not yet been considered in the context of human reasoning, such as stable models and partial stable models. To model human reasoning, it is essential that only some predicates can be subjected to closed world reasoning, while others are handled by open world reasoning. In our framework, variants of familiar logic programing semantics that are extended with this feature are derived from a generic circumscription based representation. Further, we develop a two-valued representation of a three-valued logic that renders semantics considered for human reasoning based on the Fitting operator.

## Table of Contents

## 1 Introduction

When humans are presented with reasoning tasks typical "fallacies" can be observed, conclusions that are not sound in a naive classical logic understanding. Such patterns of human reasoning have been researched quite intensely in cognitive science, with [Byr89] being a landmark work. Recently, an approach to model such patterns of human reasoning by means of logic programming has been presented [SvL05,SvL08]. It involves a two-step process, termed reasoning *to* and reasoning *from* and interpretation. The first step consists of the construction of a logic program from e.g. natural language statements and the choice of a logic programming semantics. The second step is then the actual reasoning, straightforwardly performed with respect to the program and chosen semantics.

The logic programming semantics originally suggested for this purpose in [SvL05] had in the meantime been subject to various corrections and improvements [HR09a,HR09b,HPW11]. Here, we also focus on the investigation of logic programming semantics, assuming that the programs are constructed just as described in [SvL08]. So we have given logic programs on the one hand and given

tables of empirical results about human reasoning on the other hand. The objective is to devise logic programming semantics under which the logic programs yield results that match the empirical results.

With respect to this problem, so far only fixed point characterizations of the semantics of logic programs have been considered [SvL08,HR09a,HR09b,HPW11]. In this paper, we pursue a different approach: Logic programs are represented as classical formulas, where their semantics as a logic program is expressed by wrapping them into formulas of classical logic extended by operators for circumscription and projection. Projection is a generalization of second-order quantification [Wer08]. Like circumscription, it can be processed computationally by variants of second-order quantifier elimination. In short, we map human reasoning via logic programming to a classical logic framework that could be implemented by a general classical reasoner with the ability to perform variants of second-order quantifier elimination. With this approach, we specify and compare adaptions of different well-known logic programming semantics to the modeling of human reasoning tasks. We show this here for human reasoning tasks that involve *forward* reasoning, where conditionals and truth values of antecedents are presented to the subjects, and it is investigated which values they do ascribe to consequents. Modeling human reasoning tasks that involve *backward* reasoning has also been considered in [SvL05,SvL08] and elaborated in an abductive framework [HPW11]. We do not consider backward reasoning in this paper, but a suitable variant of abduction that can be expressed in terms of circumscription and projection has been worked out and will be described in a further paper.

The paper is structured as follows: In Section 2 we introduce notation and the semantic framework, which is applied in Section 3 to model some familiar two- and three-valued semantics of logic programming, generalized by allowing to specify predicates to be handled by open world reasoning, which is necessary to model human reasoning. In Section 4 we outline the approach of [SvL05,SvL08] to model human reasoning by means of logic programs. We discuss the adequacy of the considered semantics to model empirical results obtained in cognitive science, exemplified with forward reasoning tasks that involve suppression. In the next two sections we discuss particular technical issues: Alternate ways of handling the predicates subjected to open world reasoning in Section 5, and a mapping of a three-valued logic that renders the semantics of the Fitting operator to classical two-valued logic in Section 6. We conclude in Section 7 with summarizing the contributions of this work and discussing implied issues for future research.

For the theorems in the paper, proof are given in the appendix. Propositions serve as lemmas or for illustration and are stated without proofs, since they are not hard to prove or can essentially be found in related papers [Wer08,Wer10c,Wer10a].

## 2 Notation and Semantic Framework

In this paper, we use classical *propositional* logic as basis. We extend it with operators for projection and circumscription, and develop logic programming

semantics with this extended classical logic. This framework could be straightforwardly generalized to *first-order* logic instead of propositional logic as basis, as actually shown for projection in [Wer08], for circumscription in [Wer10c] and for logic programming semantics in [Wer10a]. However, there are some particular issues related to the use of first-order quantification in the modeling of human reasoning, which will be discussed in Section 5.1.

### 2.1 Type Indicating Symbols

We use the following symbols, also with sub- and superscripts, to stand for items of types as indicated in the following table (definitions of these types are given later on), considered implicitly as universally quantified in definition, theorem and proposition statements:

$F, G, H$: formula;
$A$: atom;
$L$: literal;
$S$: scope (that is, set of literals);
$I, J$: interpretation.

### 2.2 Formulas

We consider *formulas* of classical propositional logic, extended by operators for circumscription and projection. They are constructed from propositional *atoms* (or, synonymously, 0-ary *predicates*), truth value constants $\top, \bot$, the unary connective $\neg$, binary connectives $\wedge, \vee, \rightarrow, \leftarrow, \leftrightarrow$, as usual, and the two operators $\mathsf{circ}$ and $\mathsf{project}$ to express circumscription and projection. As meta-level notation we use n-ary versions of $\wedge$ and $\vee$.

### 2.3 Predicate Groups

Semantics for knowledge representation often involve what might be described as handling different *occurrences* of a predicate differently – for example depending on whether it is subject to negation as failure. If such semantics are to be modeled with classical logic, then these occurrences can be identified by using distinguished predicates, which are equated with the original ones when required. Predicate groups are a means to express this: We assume a fixed set of atoms $\mathsf{ALL}$ that can be partitioned into a finite number of disjoint sets of equal cardinality which we call *predicate groups*. The idea is that each "original" predicate (or, for propositional logic, synonymously, atom) is replicated once in each group. The respective copy of $p$ in group $\mathsf{PG}_i$ is then written $p^i$, where $i$ is a natural number, denoting the index starting from 0 of the predicate group in some assumed fixed ordering. We use this index number also to denote the predicate group itself. Formally, the partitioning into predicate groups can be modeled by means of a total ordering on predicates such that $p$ denotes the position of $p^i$ within predicate group $i$, sorted according to that ordering. We call

such a position *ungrouped predicate*. The set of all ungrouped predicates is written $\mathcal{P}$. If $i$ and $j$ are predicate groups, we say that $p^i$ and $p^j$ are *corresponding* predicates.

### 2.4 Literals

We use literals as objects by themselves, related to formulas, but not as constituents of formulas. Thus, we always write them prefixed with a sign also if they are positive. More precisely: A *literal* is a pair of an atom and a sign. We write the positive (negative) *literal* with atom $A$ as $+A$ ($-A$). The complement of a literal $L$ is denoted by $\overline{L}$. If $S$ is a set of literals, then $\overline{S}$ denotes the set of the complements of the members of $S$. The set of literals that "occur" in a formula $F$ is denoted by $\mathcal{L}(F)$, defined formally as follows: $\mathcal{L}(A) \stackrel{\text{def}}{=} \{+A\}$ for an atom $A$; $\mathcal{L}(\top) \stackrel{\text{def}}{=} \mathcal{L}(\bot) \stackrel{\text{def}}{=} \{\}$; $\mathcal{L}(\neg F_1) \stackrel{\text{def}}{=} \overline{\mathcal{L}(F_1)}$; $\mathcal{L}(F_1 \wedge F_2) \stackrel{\text{def}}{=} \mathcal{L}(F_1 \vee F_2) \stackrel{\text{def}}{=} \mathcal{L}(F_1) \cup \mathcal{L}(F_2)$; $\mathcal{L}(F_1 \leftarrow F_2) \stackrel{\text{def}}{=} \mathcal{L}(F_1 \vee \neg F_2)$; $\mathcal{L}(F_1 \leftrightarrow F_2) \stackrel{\text{def}}{=} \mathcal{L}((F_1 \leftarrow F_2) \wedge (F_2 \leftarrow F_1))$, $\mathcal{L}(\mathsf{project}_S(F)) \stackrel{\text{def}}{=} \mathcal{L}(F)$; $\mathcal{L}(\mathsf{circ}_S(F)) \stackrel{\text{def}}{=} \mathcal{L}(F) \cup S \cup \overline{S}$. We say that a formula $F$ is *over* a set of literals $S$ if $\mathcal{L}(F) \subseteq S$.

### 2.5 Scopes

The variants of circumscription and projection that we use are parameterized by a set of literals. We call a set of literals in this context a *scope*. In the specifications of scopes we use the following shorthands: A *set of atoms* stands for the set of all literals whose atom is in the given set. For example, $\{p^0, q^1\}$ stands for $\{+p^0, -p^0, +q^1, -q^1\}$. If $S$ is a scope specifier, then $+S$ ($-S$) denotes the set of all positive (negative) literals in $S$. If $P$ denotes a set of ungrouped atoms, and $i$ a predicate group, then $P^i$ is the set of all literals whose atom is in $\{p^i \mid p \in P\}$. We write just the *natural number* $i$ as shorthand for $\mathcal{P}^i$, that is, the set of all literals whose atom is in predicate group $i$. For example, if $\mathcal{P} = \{p, q\}$, then 1 stands for $\{+p^1, -p^1, +q^1, -q^1\}$.

### 2.6 Semantic Framework

An *interpretation* is a set of literals that contains for all atoms $A$ exactly one of $+A$ or $-A$. The satisfaction relation between interpretations and formulas is defined by clauses, one for atoms and one for each logic operator, as shown for a selection of operators in the Table 1 below. Based on the satisfaction relation, entailment and equivalence are defined as usual: $F_1 \models F_2$ if and only if for all interpretations $I$ it holds that if $I \models F_1$ then $I \models F_2$. $F_1 \equiv F_2$ if and only if $F_1 \models F_2$ and $F_2 \models F_1$.

### 2.7 Projection

The formula $\mathsf{project}_S(F)$ is called the *projection* of formula $F$ onto scope $S$. It is semantically defined in Table 1. The *forgetting* in $F$ about $S$ is a variant of

**Table 1.** The Satisfaction Relation (Excerpt)

| | | |
|---|---|---|
| $I \models A$ | $\text{iff}_{\text{def}}$ | $+A \in I$. |
| $I \models \top$. | | |
| $I \models \neg F$ | $\text{iff}_{\text{def}}$ | $I \not\models F$. |
| $I \models F_1 \wedge F_2$ | $\text{iff}_{\text{def}}$ | $I \models F_1$ and $I \models F_2$. |
| $I \models \mathsf{project}_S(F)$ | $\text{iff}_{\text{def}}$ | there exists a $J$ such that: $J \models F$ and $J \cap S \subseteq I$. |
| $I \models \mathsf{circ}_S(F)$ | $\text{iff}_{\text{def}}$ | $I \models F$, and there does not exist a $J$ such that: $J \models F$ and $J \cap S \subset I \cap S$. |

projection, where the scope is considered complementary:

$$\mathsf{forget}_S(F) \stackrel{\text{def}}{=} \mathsf{project}_{\mathsf{ALL}-S}(F). \tag{i}$$

The particular variants of projection and forgetting that we use are *literal projection* and *literal forgetting* [Wer08,LLM03]. Combined with first-order logic, projection generalizes second-order quantification. Combined with propositional logic, Boolean quantification: A quantified Boolean formula $\exists p\ F$ can be expressed as $\mathsf{forget}_{\{p\}}(F)$ or, equivalently as $\mathsf{project}_{\mathsf{ALL}-\{p\}}(F)$. *Literal* projection also allows to express, so-to-speak, quantification upon just the positive or negative occurrences of a Boolean variable in a formula. Intuitively, the literal projection of a formula $F$ onto scope $S$ is a formula that expresses about literals in $S$ the same as $F$, but expresses nothing about other literals. A projection of a propositional formula is equivalent to a propositional formula (without projection operator) in which only literals in the projection scope do occur. Such a sentence is a *uniform interpolant* of the formula with respect to the scope. A naive way to construct such a sentence is indicated by the following equivalences, where $F[p\backslash\top]$ ($F[p\backslash\bot]$) denotes $F$ with all occurrences of atom $p$ replaced by $\top$ ($\bot$): (1.) $\mathsf{forget}_{\{p\}}(F) \equiv F[p\backslash\top] \vee F[p\backslash\bot]$. (2.) $\mathsf{forget}_{\{+p\}}(F) \equiv F[p\backslash\top] \vee (\neg p \wedge F[p\backslash\bot])$. (3.) $\mathsf{forget}_{\{-p\}}(F) \equiv (p \wedge F[p\backslash\top]) \vee F[p\backslash\bot]$.

In the course of this paper we will use projection for different purposes. One of them is to provide a semantic account of systematically replacing predicates from one predicate group by their correspondents from another one. Let $i, j$ be different predicate groups. We define

$$\mathsf{ren}_{i\backslash j}(F) \stackrel{\text{def}}{=} \mathsf{forget}_i(F \wedge \bigwedge_{p \in \mathcal{P}} (p^j \leftrightarrow p^i)). \tag{ii}$$

We define $\mathsf{ren}_{[i_1\backslash j_1,\ ...,\ i_n\backslash j_n]}(F)$ as a shorthand for $\mathsf{ren}_{i_n\backslash j_n}(...(\mathsf{ren}_{i_1\backslash j_1}(F))...)$. The formula $\mathsf{ren}_{i\backslash j}(F)$ is equivalent to $F$ with all occurrences of predicates from $i$ replaced by their respective corresponding predicates from $j$.

### 2.8 Scope Determined Circumscription

Like project, the circ operator has a scope specifier and a formula as arguments. It is also semantically defined in the Table 1. It allows to express variants of parallel

predicate circumscription where the effects are controlled by the scope parameter [Wer10c]. Atoms that occur just in a *positive* literal in the circumscription scope are minimized, atoms that occur just in a *negative* literal are maximized, atoms that occur in *both polarities* are fixed and atoms that do *not at all* occur in the scope are varying. Thus, if $F$ is a formula over disjoint sets of predicates $P$, $Q$ and $Z$, then the *parallel predicate circumscription of $P$ in $F$ with fixed $Q$ and varied $Z$* [Lif94], traditionally expressed as $\mathrm{CIRC}[F; P; Z]$, can be written as $\mathsf{circ}_{+P \cup Q}(F)$.

## 3 Considered Logic Programming Semantics

### 3.1 Normal Logic Programs and their Classical Representation

We consider finite normal logic programs, that is, finite sets of rules of the form

$$p \leftarrow q_1 \wedge \ldots \wedge q_n \wedge \neg r_1 \wedge \ldots \wedge \neg r_m, \tag{iii}$$

where $n, m \geq 0$ and $p, q_i, r_i$ are atoms. The logical connectives are understood there with a special meaning that depends on the associated logic programming semantics. The *classical representation of a normal logic program* is a classical propositional sentence, obtained from the program by forming the conjunction of its members and replacing each atom by its representative from the indicated group, according to the following schema:

$$p^0 \leftarrow q_1^2 \wedge \ldots \wedge q_n^2 \wedge \neg r_1^1 \wedge \ldots \wedge \neg r_m^1. \tag{iv}$$

In (iv) the logical connectives are understood in the classical sense. Information that was expressed in (iii) by the positioning of an atom in a rule head, positive body or negative body, respectively, is now captured instead by the assignment to predicate groups.

### 3.2 Open Predicates

As explicated below in Section 4.1, it is essential for the application of logic programming to model human reasoning according to the approach of [SvL08] that some of the predicates which do not occur in a head are distinguished as *open* and handled by open world reasoning instead of closed world reasoning. In [SvL08,HR09a] the syntax of logic programs is extended, such that the open predicates can be specified within the program itself: Exactly those predicates in the program that do not occur in some head are considered as open. Rules of the form $p \leftarrow \bot$ serve just for the purpose to exclude $p$ from the open predicates. The logic programming semantics that we will consider can straightforwardly be generalized to take open predicates into account. Instead of tweaking the program syntax, we characterize them by operators with two parameters, a program and a set of open predicates.

### 3.3 Two-Valued Semantics

The semantics of Clark's completion and the stable model semantics can be expressed as follows, for classical representations $F$ of logic programs, that is, formulas over scope $0 \cup 1 \cup 2$ (representations of *normal* logic programs are special cases of such formulas), and sets $O$ of ungrouped predicates such that $\mathcal{L}(F) \cap O^0 = \emptyset$ which are considered open:

$$\mathsf{comp}(F, O) \stackrel{\text{def}}{=} \mathsf{ren}_{1\setminus 0}(\mathsf{circ}_{+0\cup 1\cup O^0}(\mathsf{ren}_{2\setminus 1}(F))). \tag{v}$$

$$\mathsf{stable}(F, O) \stackrel{\text{def}}{=} \mathsf{ren}_{1\setminus 0}(\mathsf{circ}_{+0\cup 1\cup O^0}(\mathsf{ren}_{2\setminus 0}(F))). \tag{vi}$$

The characterization of stable models in terms of circumscription originates from [Lin91, Section 3.4.1] (see also [Lif08]). It is expressed here just in terms of projection (for the renaming) and circumscription. In the definition (v) it is combined with the characterization of Clark's completion in terms of stable models with negation as failure in the head [IS98]. In our classical representations of logic programs, positive body literals are assigned to predicate group 2. By the innermost renaming step in (v) and (vi) they are reassigned to 0 for the stable models semantics, or to 1 for the completion semantics, respectively.

If $F$ is the classical representation of a normal logic program, that is, a set of rules of the form (iii), then Clark's completion of the program is equivalent to $\mathsf{comp}(F, \{\})$, after dropping the predicate group superscripts. A set of atoms is an answer set of the program according to the stable model semantics if and only if it is obtained by dropping the predicate group superscripts from the set of atoms with predicate group 0 in the positive literals of some model of $\mathsf{stable}(F, \{\})$. Based on the following proposition, a proof of the correspondence of Clark's completion to the characterization in definition (v) can be sketched:

**Proposition 1 (Completion: Syntactic and via Circumscription).** *Let $F$ be a formula and let $L_1, \ldots, L_n$ be literals such that for all $i \in \{1, \ldots, n\}$ there exist formulas $G_i$ and $H_i$ with the properties (1.) $F \equiv (L_i \leftarrow G_i) \wedge H_i$, (2.) $\mathcal{L}(G_i) \cap \{L_1, \ldots, L_n, \overline{L_1}, \ldots, \overline{L_n}\} = \emptyset$, and (3.) $\mathcal{L}(H_i) \cap (L_i \cup \overline{L_i}) = \emptyset$. Then,*

$$F \wedge \bigwedge_{i=1}^{n} (L_i \rightarrow G_i) \;\equiv\; \mathsf{circ}_{\mathsf{ALL}-\overline{\{L_1,\ldots,L_n\}}}(F).$$

If $F$ is the classical representation of a normal logic program and we assume $\mathsf{ALL} = 0 \cup 1 \cup 2$, we can match the right side of the proposition with

$$\mathsf{circ}_{(0\cup 1\cup 2)-(-0)}(\mathsf{ren}_{2\setminus 1}(F)). \tag{vii}$$

The scope of the circumscription in this formula is then equal to $+0 \cup 1 \cup 2$. Since $\mathsf{ren}_{2\setminus 1}(F)$ does not express anything about predicate group 2, it can be dropped from the circumscription scope. That is, from $\mathsf{ren}_{2\setminus 1}(F) \equiv \mathsf{project}_{0\cup 1}(\mathsf{ren}_{2\setminus 1}(F))$ it can be derived that formula (vii) is equivalent to $\mathsf{circ}_{+0\cup 1}(\mathsf{ren}_{2\setminus 1}(F))$, which matches the argument of the outer renaming in the definition (v). Detailed formal justifications of the correspondences of first-order generalizations of definitions

(v) and (vi) to the respective logic programming semantics rendered by them are provided in [Wer10b].

As an example, consider the program $\{p \leftarrow p,\ q \leftarrow \neg p\}$. Its classical representation is $F \stackrel{\text{def}}{=} (p^0 \leftarrow p^2) \wedge (q^0 \leftarrow \neg p^1)$. Its Clark's completion is rendered as

$$\begin{aligned} \mathsf{comp}(F, \{\}) &\equiv \mathsf{ren}_{1\setminus 0}(\mathsf{circ}_{+0\cup 1}((p^0 \leftarrow p^1) \wedge (q^0 \leftarrow \neg p^1))) \\ &\equiv \mathsf{ren}_{1\setminus 0}((\neg p^0 \wedge \neg p^1 \wedge q^0) \vee (p^0 \wedge p^1 \wedge \neg q^0)) \\ &\equiv (\neg p^0 \wedge q^0) \vee (p^0 \wedge \neg q^0). \end{aligned} \tag{viii}$$

The stable models semantics of F is rendered as

$$\begin{aligned} \mathsf{stable}(F, \{\}) &\equiv \mathsf{ren}_{1\setminus 0}(\mathsf{circ}_{+0\cup 1}((p^0 \leftarrow p^0) \wedge (q^0 \leftarrow \neg p^1))) \\ &\equiv \mathsf{ren}_{1\setminus 0}((\neg p^0 \wedge \neg p^1 \wedge q^0) \vee (\neg p^0 \wedge p^1 \wedge \neg q^0)) \\ &\equiv \neg p^0 \wedge q^0. \end{aligned} \tag{ix}$$

The models of the logic program according to the respective semantics are then represented by the models in the sense of classical logic of the formulas $\mathsf{comp}(F, \{\})$ and $\mathsf{stable}(F, \{\})$, respectively.

### 3.4 Three-Valued Semantics

In the specifications of two-valued logic programming semantics, predicate groups serve to indicate the effect of circumscription. Two predicate groups can also be applied to express truth values $\mathsf{F}, \mathsf{U}, \mathsf{T}$ of a three-valued logic in a two-valued logic. An interpretation $I$ over (possibly a superset of) $0 \cup 1$ is said to *assign* to an atom $p$ a three-valued truth value as specified in the following table:

**Table 2.** Correspondence to Three-Valued Truth Values

| | | |
|---|---|---|
| $I$ assigns to $p$ the value $\mathsf{F}$ | iff | $I \models \neg p^0 \wedge \neg p^1$; |
| $I$ assigns to $p$ the value $\mathsf{U}$ | iff | $I \models \neg p^0 \wedge p^1$; |
| $I$ assigns to $p$ the value $\mathsf{T}$ | iff | $I \models p^0 \wedge p^1$. |

The remaining possibility that $I \models p^0 \wedge \neg p^1$ is not considered as representing a three-valued truth value. The axiom $\mathsf{cons}$ can be used to exclude models with such assignments:

$$\mathsf{cons} \stackrel{\text{def}}{=} \bigwedge_{p \in \mathcal{P}} (p^1 \leftarrow p^0). \tag{x}$$

An interpretation is called *less-or-equal-than* another interpretation if and only if each atom assigned to $\mathsf{F}$ or $\mathsf{T}$ by the first interpretation is assigned to the same three-valued truth value by the second interpretation. Models that are *least* with respect to this relation can be expressed by circumscription: If the models of $F$ satisfy $\mathsf{cons}$, then the least models of $F$ are the models of $\mathsf{circ}_{+0\cup -1}(F)$. Accordingly we define:

$$\mathsf{least}(F) \stackrel{\text{def}}{=} \mathsf{circ}_{+0\cup -1}(F). \tag{xi}$$

If cons is used together with least, it can equivalently be placed inside or outside of the least operator:

**Proposition 2 (Consistent Inside and Outside Least).**

$$\mathsf{least}(\mathsf{cons} \wedge F) \equiv \mathsf{cons} \wedge \mathsf{least}(F).$$

The following definition renders the models obtained by the Fitting operator [Fit85], and with the $O$ parameter also the so-called weak completion semantics of [HR09a]. It is defined for the same classes of arguments $F$ and $O$ as comp and stable.

$$\begin{aligned}\mathsf{fitting}(F,O) \stackrel{\text{def}}{=}\ & \mathsf{cons} \wedge \\ & \mathsf{ren}_{2\backslash 0}(F) \wedge \\ & \mathsf{ren}_{[0\backslash 2,\ 1\backslash 0,\ 2\backslash 1]}(\mathsf{forget}_{+0}(\mathsf{circ}_{+0\cup 2\cup 1\cup O^0}(F))).\end{aligned} \tag{xii}$$

If $F$ is the classical representation of a normal logic program, then $\mathsf{least}(\mathsf{fitting}(F,\{\}))$ has a single model that represents the least three-valued model of the Fitting operator by combinations of two-valued values in predicate groups 0 and 1, as specified above. As shown in [Fit85], the least model of the Fitting operator is also the least model of the completion of the program, considered as a formula in a certain three-valued logic. With Theorem 3 in Section 6 we show the correspondence of the latter to $\mathsf{least}(\mathsf{fitting}(F,\{\}))$, which then obviously implies that $\mathsf{least}(\mathsf{fitting}(F,\{\}))$ correctly renders the least model obtained with the Fitting operator.

As an example, consider the program $\{p \leftarrow p,\ q \leftarrow \neg p,\ r \leftarrow \neg r\}$. Its classical representation is $F \stackrel{\text{def}}{=} (p^0 \leftarrow p^2) \wedge (q^0 \leftarrow \neg p^1) \wedge (r^0 \leftarrow \neg r^1)$.

$$\begin{aligned}
&(1) && \mathsf{least}(\mathsf{fitting}(F,\{\})) \\
&(2) && \equiv \mathsf{least}(\mathsf{cons} \wedge \\
&(3) && \qquad (p^0 \leftarrow p^0) \wedge (q^0 \leftarrow \neg p^1) \wedge (r^0 \leftarrow \neg r^1) \wedge \\
&(4) && \qquad \mathsf{ren}_{[0\backslash 2,\ 1\backslash 0,\ 2\backslash 1]} \\
&(5) && \qquad\quad (\mathsf{forget}_{+0}((p^0 \leftrightarrow p^2) \wedge (q^0 \leftrightarrow \neg p^1) \wedge (r^0 \leftrightarrow \neg r^1)))) \\
&(6) && \equiv \mathsf{least}(\mathsf{cons} \wedge \\
&(7) && \qquad (p^0 \leftarrow p^0) \wedge (q^0 \leftarrow \neg p^1) \wedge (r^0 \leftarrow \neg r^1) \wedge \\
&(8) && \qquad (p^1 \rightarrow p^1) \wedge (q^1 \rightarrow \neg p^0) \wedge (r^1 \rightarrow \neg r^0)))) \\
&(9) && \equiv \mathsf{least}((p^0 \wedge p^1 \wedge \neg q^0 \wedge \neg q^1 \wedge \neg r^0 \wedge r^1) \vee \\
&(10) && \qquad (\neg p^0 \wedge p^1 \wedge q^0 \wedge q^1 \wedge \neg r^0 \wedge r^1) \vee \\
&(11) && \qquad (\neg p^0 \wedge p^1 \wedge \neg q^0 \wedge q^1 \wedge \neg r^0 \wedge r^1) \vee \\
&(12) && \qquad (\neg p^0 \wedge p^1 \wedge \neg q^0 \wedge \neg q^1 \wedge \neg r^0 \wedge r^1) \vee \\
&(13) && \qquad (\neg p^0 \wedge \neg p^1 \wedge q^0 \wedge q^1 \wedge \neg r^0 \wedge r^1)) \\
&(14) && \equiv \neg p^0 \wedge p^1 \wedge \neg q^0 \wedge q^1 \wedge \neg r^0 \wedge r^1.
\end{aligned} \tag{xiii}$$

In line (4) the circumscription in the definition of fitting has been computed, yielding the completion of $F$. Line (8) shows line (4)–(5) after applying forgetting to extract just the converse implications of the completion and renaming. The extraction of the converse implications requires that *literal* forgetting which distinguishes polarity has to be applied, and is justified by the following proposition:

**Proposition 3 (Semantic Extraction of the Completion Addendum).** *Let $L_1, \ldots, L_n$, $G_1, \ldots G_n$, where $n \geq 0$, be literals and formulas, respectively, such that for all $i \in \{1, \ldots, n\}$ it holds that $\mathcal{L}(G_i) \cap \{L_1, \ldots, L_n, \overline{L_1}, \ldots, \overline{L_n}\} = \emptyset$. Then*

$$\mathsf{forget}_{\{L_1,\ldots,L_n\}}(\bigwedge_{i=1}^{n}(L_i \leftrightarrow G_i)) \equiv \bigwedge_{i=1}^{n}(L_i \rightarrow G_i).$$

Lines (7) and (8) in the example (xiii) show that the core of the characterization of the semantics of the Fitting operator could be specified syntactically as a program translation that consists of the original implications with the same predicate group settings as for the stable models semantics conjoined with the converse implications obtained from the completion, but with flipped predicate groups. In lines (9)–(13) the argument of least is shown in a disjunctive normal form where each model corresponds to a conjunction. The corresponding three valued models written as pairs of the atoms assigned to T and the atoms assigned to F are as follows: $\langle\{p\}, \{q\}\rangle$ for the conjunction in (9); $\langle\{q\}, \{\}\rangle$ for (10); $\langle\{\}, \{\}\rangle$ for (11); $\langle\{\}, \{q\}\rangle$ for (12); $\langle\{q\}, \{p\}\rangle$ for (13). Line (14) shows the result of restricting to the least models, that is, circumscribing upon $+0 \cup -1$. Correspondingly, line (14) represents $\langle\{\}, \{\}\rangle$.

Formula partial-stable$(F, O)$ defined below renders the partial stable models semantics [Prz90a,Prz90b]. We follow the approach of [JNS$^+$06] where a translation of is specified such that the stable models of the translated program correspond to the partial stable models of the original program. This translation has its roots in [Sch95]. The definition of partial-stable combines the translation of [JNS$^+$06], where three-valued interpretations are represented by two-valued interpretations according to Table 2, with the characterization of stable models according to the definition (vi). Each of these two translations involves discrimination between two predicate groups. The combination of both yields four predicate groups, which are reduced in the final value of partial-stable by renaming to groups 0 and 1. Formulas partial-stable$(F, O)$ are defined for the same classes of arguments $F$ and $O$ as for the other considered semantics. The models of partial-stable$(F, O)$ represent the three-valued partial stable models by combining the values of atoms for predicate groups 0 and 1. The $O$ parameter generalizes the partial stable models semantics by allowing to handle predicates in a given set specially as open.

$$\begin{aligned}\mathsf{partial\text{-}stable}(F, O) \stackrel{\text{def}}{=} \mathsf{ren}_{[2\backslash 0, 3\backslash 1]}(&\\ \mathsf{circ}_{+0\cup+1\cup 2\cup 3\cup O^0\cup O^1}(&\mathsf{cons} \wedge\\ &\mathsf{ren}_{[2\backslash 0, 1\backslash 3]}(F) \wedge\\ &\mathsf{ren}_{[1\backslash 3, 0\backslash 1, 2\backslash 1, 3\backslash 2]}(F))).\end{aligned} \tag{xiv}$$

That partial-stable$(F, \{\})$ correctly renders the partial stable models semantics follows from the correctness of the encoding of that semantics into the stable models semantics, shown in [JNS$^+$06], and the correctness of the characterization of the stable models semantics as given in definition (vi).

We illustrate partial-stable with the same example as fitting, the program $\{p \leftarrow p,\ q \leftarrow \neg p,\ r \leftarrow \neg r\}$ whose classical representation is $F \stackrel{\text{def}}{=} (p^0 \leftarrow$

$p^2) \wedge (q^0 \leftarrow \neg p^1) \wedge (r^0 \leftarrow \neg r^1)$. We write the superscripts that indicate the predicate groups as binary numbers with two digits:

$$
\begin{array}{ll}
(1) & \quad \mathsf{partial\text{-}stable}(F, \{\}) \\
(2) & \equiv \mathsf{ren}_{[10\backslash 00, 11\backslash 01]}( \\
(3) & \quad\quad \mathsf{circ}_{00^+ \cup 01^+ \cup 10 \cup 11}(\bigwedge_{p^{00} \in \mathsf{PG}_{00}} (p^{01} \leftarrow p^{00}) \wedge \\
(4) & \quad\quad\quad\quad (p^{00} \leftarrow p^{00}) \wedge (q^{00} \leftarrow \neg p^{11}) \wedge (r^{00} \leftarrow \neg r^{11}) \wedge \\
(5) & \quad\quad\quad\quad (p^{01} \leftarrow p^{01}) \wedge (q^{01} \leftarrow \neg p^{10}) \wedge (r^{01} \leftarrow \neg r^{10}))) \\
(6) & \equiv \neg p^{00} \wedge \neg p^{01} \wedge q^{00} \wedge q^{01} \wedge \neg r^{00} \wedge r^{01}.
\end{array}
\tag{xv}
$$

The binary notation of predicate group superscripts indicates how the translation of [JNS+06] into stable models and representation of stable models according to definition (vi) are combined: The right digit corresponds to the group discrimination required by the translation into stable models, the left digit to the discrimination required by expressing the stable models semantics with circumscription. The three-valued model represented by line (6) is $\langle \{q\}, \{p\} \rangle$.

# 4 Adequacy for the Suppression Task with Forward Reasoning

## 4.1 Modeling Human Reasoning by Nonmonotonic Logics According to Stenning and van Lambalgen

In the field of human reasoning, it is observed that humans *suppress* certain inferences, that is, do not draw certain conclusions from given facts and conditionals. This comprises conclusions that would be valid as well as conclusions that would not be valid in a straightforward classical logic reading of the natural language sentences presented to the subjects. Observed reasoning by humans is inherently nonmonotonic: the presence of an additional sentence can effect suppression of a conclusion that would have been drawn without the additional sentence. Moreover, whether certain inferences are suppressed can not be determined just by the syntactic form of the natural language sentences. Corresponding experiments have been considered as evidence that logic is inadequate to model human reasoning [Byr89]. Stenning and van Lambalgen [SvL05,SvL08] propose an approach based on nonmonotonic logic that is adequate for modeling the reported experiments.

Instead of a naive translation of natural language sentences into classical logic, they assume a two stage process, where the first stage, *reasoning to an interpretation*, is an "interpretative" process that assigns logical form which is not implied by the syntactical structure of the given natural language sentences, but takes contextual information and background knowledge into account. This stage includes the understanding of conditionals as implicitly relativized with a precondition that excludes abnormal situations and the decision to apply a specific logic, such as determining that nonmonotonic closed world reasoning should be applied with respect to abnormality predicates and certain other predicates. The

second stage, *reasoning from an interpretation*, is then considered as straightforward reasoning with respect to the logical representations of conditionals and facts resulting from the previous stage.

Reasoning tasks investigated with respect to suppression fall into two categories: First, *forward* reasoning tasks, where the truth value of the antecedent of a conditional is given and it is to be determined whether a consequent holds. Depending on the truth value of the antecedent, the forward reasoning patterns are called *modus ponens* and *denial of the antecedent*. Second, *Backward reasoning*, where values of a consequent are given and it is to be determined whether an antecedent holds. Here we just consider forward reasoning tasks.

In the presentation of the human reasoning tasks from [Byr89] we follow [HPW11]. For the text phrases given to the subjects we introduce the following abbreviations:

**Table 3.** Conditionals and Facts used in the Experiments

| | |
|---|---|
| $C_e$ | *If she has an essay to write she will study late in the library.* |
| $C_t$ | *If she has a textbook to read she will study late in the library.* |
| $C_o$ | *If the library stays open she will study late in the library.* |
| $e$ | *She has an essay to write.* |
| $l$ | *She will study late in the library.* |
| $o$ | *If the library stays open she will study late in the library.* |
| $t$ | *She has textbooks to read.* |

We abbreviate the negation of a fact $X$ by $\neg X$, e.g. $\neg e$ denotes *she does not have an essay to write*. The empirical results reported originally in [Byr89] and obtained similarly in a repetition of the experiment in [DSSd00] are shown in Table 4: Phrases $K$ presented to the subjects and the proportion of subjects according to [Byr89] and [DSSd00] who have drawn conclusion $Q$ from them. Tasks 1–3 are *modus ponens* inferences, and 4–6 *denial of the antecedent* inferences. In both of these groups three experiments have been made: First (tasks 1 and 4), the single conditional $C_e$ is given, along with a statement that its precondition $e$ is true or false, respectively. Second (tasks 2 and 5), the setting of the first variant is extended by a further conditional $C_t$ that is typically understood as providing an *alternative* possible reason for the consequent $l$. Third (tasks 3 and 6), the setting of the first task is extended by a conditional $C_o$ that in fact represents an *additional* necessary precondition for the consequent $l$, although it is syntactically of the same form as $C_t$. As Table 4 shows, the addition of $C_o$ to the setting of task 1 effects that the conclusion of $l$ is suppressed by many subjects: In the experiment reported in [Byr89], only 38% of the subjects concluded $l$ in task 3. In the experiment repeated by [DSSd00], this number is larger (61%) but still significantly smaller compared to tasks 1 (88%) and 2 (93%).
According to the approach of Stenning and van Lambalgen, logic programs can model the empirical results shown in Table 4. Together with some chosen logic programming semantics, they form the result of the *reasoning to an interpre-*

**Table 4.** Empirical Results about Human Reasoning Tasks

| | $K$ | $Q$ | [Byr89] | [DSSd00] |
|---|---|---|---|---|
| 1. | $C_e, e$ | $l$ | 96% | 88% |
| 2. | $C_e, C_t, e$ | $l$ | 96% | 93% |
| 3. | $C_e, C_o, e$ | $l$ | 38% | 61% |
| 4. | $C_e, \neg e$ | $\neg l$ | 46% | 49% |
| 5. | $C_e, C_t, \neg e$ | $\neg l$ | 4% | 22% |
| 6. | $C_e, C_o, \neg e$ | $\neg l$ | 63% | 49% |

**Table 5.** Logic Programs Corresponding to the Tasks in Table 4

$P_1 \stackrel{\text{def}}{=} \{l \leftarrow e \wedge \neg ab_1,\ ab \leftarrow \bot,\ e \leftarrow \top\}$. $O_1 \stackrel{\text{def}}{=} \{\}$.
$P_2 \stackrel{\text{def}}{=} \{l \leftarrow e \wedge \neg ab_1,\ ab_1 \leftarrow \bot,\ l \leftarrow t \wedge \neg ab_2,\ ab_2 \leftarrow \bot,\ e \leftarrow \top\}$. $O_2 \stackrel{\text{def}}{=} \{t\}$.
$P_3 \stackrel{\text{def}}{=} \{l \leftarrow e \wedge \neg ab_1,\ ab_1 \leftarrow \neg o,\ l \leftarrow o \wedge \neg ab_3,\ ab_3 \leftarrow \neg e,\ e \leftarrow \top\}$. $O_3 \stackrel{\text{def}}{=} \{o\}$.
$P_4 \stackrel{\text{def}}{=} \{l \leftarrow e \wedge \neg ab_1,\ ab \leftarrow \bot,\ e \leftarrow \bot\}$. $O_4 \stackrel{\text{def}}{=} \{\}$.
$P_5 \stackrel{\text{def}}{=} \{l \leftarrow e \wedge \neg ab_1,\ ab_1 \leftarrow \bot,\ l \leftarrow t \wedge \neg ab_2,\ ab_2 \leftarrow \bot,\ e \leftarrow \bot\}$. $O_5 \stackrel{\text{def}}{=} \{t\}$.
$P_6 \stackrel{\text{def}}{=} \{l \leftarrow e \wedge \neg ab_1,\ ab_1 \leftarrow \neg o,\ l \leftarrow o \wedge \neg ab_3,\ ab_3 \leftarrow \neg e,\ e \leftarrow \bot\}$. $O_6 \stackrel{\text{def}}{=} \{o\}$.

*tation* phase. The semantics considered in [SvL08] is a three-valued completion semantics, a variant of the Fitting operator semantics [Fit85]. The presentation in [SvL05,SvL08] erroneously associates the wrong three-valued logic with programs, which has been corrected in [HR09a], where this semantics has been termed *weak completion semantics*.

Since we are here interested in considering different three- and also two-valued logic programming semantics we do not commit to a particular one, but emphasize a particular required feature that can actually be combined with various logic programming semantics: The semantics needs to supports that *some* predicates are handled by closed world reasoning, while others are handled with open world reasoning. This can technically be expressed in different ways, as discussed in Sections 3.2 and 5. In our framework, we express it by specifying semantics for logic programs with respect to a second parameter besides the program, the set of predicates that are to be considered as open, that is, to be handled with open world reasoning, where all other predicates are to be handled with closed world reasoning.

In Table 5, for each task $i$ from Table 4, the corresponding the logic program $P_i$ and set $O_i$ of open predicates is shown. Here we use conventional notation following the schema (iii). To remain faithful to the source literature [SvL08,HR09a,HPW11] we keep rules with $\bot$ as body, which are redundant in presence of the explicitly given sets of open predicates. The reasoning *to* an interpretation described in [SvL05,SvL08] which, given the combinations of natural language conditionals and facts from Table 4, yields these programs can be summarized in the following steps:

1. Consider a conditional "if A then B" as standing for "if A and nothing abnormal is the case, then B". This leads to the representation of $C_e$ as $l \leftarrow e \wedge \neg ab$, of $C_t$ as $l \leftarrow t \wedge \neg ab_2$, and of $C_o$ as $l \leftarrow o \wedge \neg ab_3$.

**Table 6.** Models of the Programs in Table 5 Obtained with Different Semantics

| $P$ $O$ | *3-valued* | *2-valued* | *3-valued* $O=\{\}$ | *2-valued* $O=\{\}$ |
|---|---|---|---|---|
| $P_1$ $O_1$ | $\langle\{e,l\},\{ab_1\}\rangle$ | $\{e,l\}$ | $\langle\{e,l\},\{ab_1\}\rangle$ | $\{e,l\}$ |
| $P_2$ $O_2$ | $\langle\{e,l\},\{ab_1,ab_2\}\rangle$ | $\{e,l\},\{e,l,t\}$ | $\langle\{e,l\},\{ab_1,ab_2,t\}\rangle$ | $\{e,l\}$ |
| $P_3$ $O_3$ | $\langle\{e\},\{ab_3\}\rangle$ | $\{e,l,o\},\{ab_1,e\}$ | $\langle\{ab_1,e\},\{ab_3,l,o\}\rangle$ | $\{ab_1,e\}$ |
| $P_4$ $O_4$ | $\langle\{\},\{ab_1,e,l\}\rangle$ | $\{\}$ | $\langle\{\},\{ab_1,e,l\}\rangle$ | $\{\}$ |
| $P_5$ $O_5$ | $\langle\{\},\{ab_1,ab_2,e\}\rangle$ | $\{\},\ \{l,t\}$ | $\underline{\langle\{\},\{ab_1,ab_2,e,l,t\}\rangle}$ | $\underline{\{\}}$ |
| $P_6$ $O_6$ | $\langle\{ab_3\},\{e,l\}\rangle$ | $\{ab_3,o\},\{ab_1,ab_3\}$ | $\overline{\langle\{ab_1,ab_3\},\{e,l,o\}\rangle}$ | $\overline{\{ab_1,ab_3\}}$ |

2. Consider all abnormality predicates, $ab_1$, $ab_2$, $ab_3$ in the examples, as subject to closed world reasoning.
3. Consider all predicates that occur as consequent in a rule as subject to closed world reasoning.
4. Consider all remaining predicates, that is, predicates that just occur in antecedents and are not abnormality predicates, as subject to open world reasoning. Since we assume a logic programming semantics in which predicates not explicitly specified as open are handled by closed world reasoning, this leads to the sets $O_i$ of open predicates shown in Table 5.
5. If there are two conditionals with the same conclusion, determine whether the premise of the second conditional is an *alternative* to the first one, like $t$ in $C_t$ which is an alternative to $e$ in $C_e$ for concluding $l$, or is *additional* to the first one, like $o$ in $C_o$. This step requires to take contextual information and background knowledge into account.
6. If the second conditional provides an *additional* premise, it "highlights the possibility of abnormality", which is expressed by adding a clause that asserts the abnormality precondition of the first conditional if the premise of the second conditional fails: $ab_1 \leftarrow \neg o$, in the example. For reasons of symmetry an analogous clause is added for the first conditional: $ab_3 \leftarrow \neg e$, in the example.
7. Add expressions for the facts: Positive facts with explicit clauses, like $e \leftarrow \top$; negative facts in a way such that they are implied under closed world reasoning. The latter is achieved by not letting negative facts be members of the set of open predicates, or in the original notation of [SvL08] by adding clauses like $e \leftarrow \bot$.

### 4.2 Adequacy of Different Logic Programming Semantics

When applied to model human reasoning, logic programming semantics should correspond in some way to empirical results obtained with human subjects. The investigations of suppression center around facts that are concluded from given conditionals and facts. A program that represents the outcome of reasoning towards an interpretation with respect to a human reasoning task should then have the same facts as consequences that are concluded by most human subjects when presented with the task. Table 6 shows the models for the programs

given in Table 5 with respect to several semantics. Column "3-*valued*" shows the three-valued models of $\mathsf{least}(\mathsf{fitting}(P_i, O_i))$ and $\mathsf{least}(\mathsf{partial\text{-}stable}(P_i, O_i))$, which happen to be the same for the given programs and sets of open predicates. The first of these formulas, $\mathsf{least}(\mathsf{fitting}(P_i, O_i))$, expresses the so-called *weak completion semantics*, which has been previously considered in investigations of adequacy for human reasoning [SvL08,HR09a,HR09b,HPW11]. It is conventionally characterized as a least fixed point semantics with respect to a variant of the Fitting operator that takes open predicates especially into account. In the table, three-valued models are written as pairs of the atoms assigned to $\mathsf{T}$ and the atoms assigned to $\mathsf{F}$. Column "2-*valued*" shows the models of $\mathsf{comp}(P_i, O_i)$ and $\mathsf{stable}(P_i, O_i)$, which again happen to be the same. These two-valued models are written as the list of the atoms that are satisfied. For $P_2$, $P_3$, $P_5$ and $P_6$, these semantics yield two models.

If we accept that a positive or negative fact is considered a *consequence* if and only if it is satisfied by all models of the program under the respective semantics, then the table yields – in coincidence for the 3- as well as the 2-valued semantics – the fact $l$ as a consequence for task 1 and 2, and $\neg l$ for task 4 and 5. For tasks 3 and 4 neither $l$ nor $\neg l$ is a consequence, due to the assignment of $l$ as $\mathsf{U}$ by the 3-valued semantics and alternate models with different truth values of $l$ by the 2-valued semantics, respectively.

Let us compare this with the empiric results in Table 4. In tasks 1 and 2 the vast majority of subjects concludes $l$, establishing the adequacy of the considered logic programming semantics with respect to these two tasks. In task 3 the conclusion of $l$ is suppressed by a large proportion of subjects. The considered logic programming semantics match this in that $l$ is not a consequence. With respect to the results reported by [Byr89], the logic programming semantics render the reasoning of the majority of subjects, with respect to [DSSd00], only of a large minority. In tasks 4, about half of the subjects conclude $\neg l$, and in task 6 about two thirds [Byr89] or also half of the subjects [DSSd00]. The considered logic programming semantics have $\neg l$ as a conclusion for this tasks, thus modeling the reasoning of this half of the subjects. In tasks 5, the conclusion of $\neg l$ is suppressed by the vast majority of subjects. In accord with this, the considered logic programming semantics match in that $\neg l$ is not a consequence.

To sum up, the considered logic programming semantics are adequate to some degree: For modus ponens tasks, conclusions are either drawn by a vast majority of subjects or suppressed by a significant number. In the first case, the logic programming semantics have the conclusions as a consequence, in the latter case not, reflecting the suppression. For denial of antecedent tasks, conclusions are either drawn by about half of the subjects or suppressed by a vast number. Again, in the first case the conclusions are consequences of the program, while in the suppression case they are not consequences.

The models obtained with two further logic programming semantics are shown in the two rightmost columns "3-*valued* $O = \{\}$" and "2-*valued* $O = \{\}$" show the values of $\mathsf{least}(\mathsf{fitting}(P_i, \{\}))$ as well as $\mathsf{least}(\mathsf{partial\text{-}stable}(P_i, \{\}))$, and $\mathsf{comp}(P_i, \{\})$ as well as $\mathsf{stable}(P_i, \{\})$, respectively. These correspond to the re-

spective semantics when open predicates are not specially taken into account. For task 5 they have $\neg l$ as a consequence, contradicting the corresponding empiric results such that these semantics can be clearly regarded as inadequate.

## 5 Ways to Express Open Predicates

As explained in Section 4.1, logic programming semantics that are applied to model human reasoning according to the approach of [SvL08] require that only some predicates are handled "by closed world reasoning", and others are to be considered as open. There are different ways to specify and technically handle these two sets of predicates.

### 5.1 First-Order Issues

As already described in Section 3.2, the way taken in [SvL08,HR09a] is to extend the syntax of logic programs such that exactly those predicates in the program that do not occur in some head are considered as open, where, otherwise redundant rules of the form $p \leftarrow \bot$ serve to exclude $p$ from the open predicates. Unfortunately, this approach leads to an ambiguity if lifted to first-order programs. Consider the following example program:

$$\{p(a) \leftarrow \top,\ q \leftarrow p(b)\}. \tag{xvi}$$

The first step in computing Clark's completion of this program, considering $p$ as a unary predicate, would result in the following program, considered equivalent to (xvi):

$$\{p(X) \leftarrow X = a,\ q \leftarrow p(b)\}. \tag{xvii}$$

The ground expansion of $(xvii)$ with respect to the Herbrand base $\{a, b\}$ is:

$$\{p(a) \leftarrow a = a,\ p(b) \leftarrow b = a,\ q \leftarrow p(b)\}. \tag{xviii}$$

Now, whether the ground atom $p(b)$ is to be considered as open depends on whether (xvi) or (xviii) is taken as basis to check whether $p(b)$ occurs in a head: With respect to (xvi), the atom $p(b)$ does not occur in a head and thus has to be considered as open. If (xviii) is taken as basis, no instance of the unary predicate $p$ is considered open.

First-order versions of the operators comp, stable, fitting and partial-stable [Wer10a] take as second argument a set of ground atoms that actually specify which *ground atoms* are to be considered as open. If a "whole predicate" is to be considered as open, this is expressed by a set containing all ground atoms with the respective predicate. Thus, from a first-order perspective, these operators permit to specify explicitly for each ground atom whether it should be considered as open.

### 5.2 Encoding Open Predicates in Standard Semantics

The logic operators that represent logic programming semantics, comp, stable, fitting and partial-stable allow to specify open predicates with a second argument, a set of predicates. It this is the empty set, then they render the "standard" versions of the respective semantics, Clark's completion, stable models semantics, Fitting operator semantics, and partial stable models semantics, respectively. In the definitions of all of these logic operators, the second argument that specifies the open predicates is passed as set of fixed predicates to circumscription.

If the logic program represented by the argument formula to one of these semantic operators corresponds to a normal logic program, then it is also possible to encode the handling of open predicates *into the program* and the semantic operator with just the empty set of open predicates. The original program with respect to a given set of open predicates is then equivalent to the modified program with respect to the empty set of open predicates. In this way, stock processors for logic programs which usually do not provide explicit support for open predicates can be applied. Let $O$ be the set of ungrouped predicates to be considered as open. For comp and fitting, such a program modification is achieved by adding $\{p \leftarrow p \mid p \in O\}$ to the program. For stable and partial-stable, such a program modification is achieved by adding $\{p \leftarrow \neg not\_p, not\_p \leftarrow \neg p \mid p \in O\}$ to the program, where the $not\_p$ are "fresh" ungrouped predicates, that is, not occurring in the original program, different for each $p \in O$. These auxiliary predicates can be eliminated from the final result by forgetting. The following two propositions formally state the correspondence of these encodings of open predicates for the considered logic programming semantics:

**Theorem 1 (Encoding Open Predicates with Completion).** *Let $F$ be the classical representation of a normal program and let $O$ be a set of ungrouped predicates such that $\mathcal{L}(F) \cap O^0 = \emptyset$. Define*

$$E \stackrel{\text{def}}{=} \bigwedge_{p \in O} (p^0 \leftarrow p^2).$$

*It then holds that:*

(i) $\mathsf{comp}(F \wedge E, \{\}) \equiv \mathsf{comp}(F, O)$.
(ii) $\mathsf{fitting}(F \wedge E, \{\}) \equiv \mathsf{fitting}(F, O)$.

**Theorem 2 (Encoding Open Predicates with Stable Models).** *Let $F$ be the classical representation of a normal program and let $O$ be a set of ungrouped predicates such that $\mathcal{L}(F) \cap O^0 = \emptyset$. Assume that for each $p \in O$ there exists a distinguished ungrouped predicate $not\_p$ that is neither in $O$ nor occurs in $F$. Define $N \stackrel{\text{def}}{=} \{not\_p \mid p \in O\}$. Define*

$$E \stackrel{\text{def}}{=} \bigwedge_{p \in O} ((p^0 \leftarrow \neg not\_p^1) \wedge (not\_p^0 \leftarrow \neg p^1)).$$

*It then holds that:*

(iii) $\mathsf{forget}_{N^0}(\mathsf{stable}(F \wedge E, \{\})) \equiv \mathsf{forget}_{N^0}(\mathsf{stable}(F, O))$.
(iv) $\mathsf{forget}_{N^0 \cup N^1}(\mathsf{pstable}(F \wedge E, \{\})) \equiv \mathsf{forget}_{N^0 \cup N^1}(\mathsf{pstable}(F, O))$.

# 6 Representing a Three-Valued Logic for Completion Semantics by a Two-Valued Logic

An assignment of three-valued truth values to *atomic* formulas, by two-valued interpretations over two predicate groups 0 and 1, has been specified in Table 2 (p. 9). This can be extended to *complex* formulas of certain three-valued logics. We develop such an extension for a particular three-valued logic that can be applied to render the logic programming semantics of the Fitting operator. Its correspondence to $\mathsf{fitting}$, defined in (xii), is shown formally as Theorem 3.

## 6.1 Representing $\mathsf{S_3}$ in Two-Valued Logic with Predicate Groups

We need some additional terminology: A *formula* in a three-valued logic is constructed from ungrouped propositional atoms and three-valued versions of the logic operators. For these, we use the same symbols as for two-valued logics, with exception of implication and biconditional where we consider variants with different three-valued semantics, identified by sub- and superscripts. We assume the two predicate groups 0 and 1. The semantics of three-valued formulas is then specified by means of a *three-valued valuation function*, a function that maps an interpretation that satisfies $\mathsf{cons}$ and a three-valued formula to a three-valued truth value. We write the interpretation argument as subscript to the function name. An interpretation $I$ is a *model* of a formula $F$ in the three-valued logic represented by some valuation function *val* if and only if $I$ satisfies $\mathsf{cons}$ and $val_I(F)$ is the so-called *designated truth value*, which is just $\mathsf{T}$, as far as we consider three-valued logics here. We now focus on a particular three-valued logic, specified by the valuation function $\mathsf{val}$, defined as follows, where the three-valued truth values are assumed to be ordered by $\mathsf{T} > \mathsf{U} > \mathsf{F}$:

$$
\begin{aligned}
\mathsf{val}_I(A) &\stackrel{\text{def}}{=} \mathsf{F} \text{ if } I \models \neg A^0 \wedge \neg A^1;\\
&\quad\ \mathsf{U} \text{ if } I \models \neg A^0 \wedge A^1;\\
&\quad\ \mathsf{T} \text{ if } I \models A^0 \wedge A^1.\\
\mathsf{val}_I(\bot) &\stackrel{\text{def}}{=} \mathsf{F}.\\
\mathsf{val}_I(\top) &\stackrel{\text{def}}{=} \mathsf{T}.\\
\mathsf{val}_I(\neg F) &\stackrel{\text{def}}{=} \neg\mathsf{val}_I(F), \text{ where } \neg\mathsf{F} = \mathsf{T},\ \neg\mathsf{U} = \mathsf{U},\ \neg\mathsf{T} = \mathsf{F}.\\
\mathsf{val}_I(F \wedge G) &\stackrel{\text{def}}{=} \mathsf{min}(\mathsf{val}_I(F), \mathsf{val}_I(G)).\\
\mathsf{val}_I(F \vee G) &\stackrel{\text{def}}{=} \mathsf{max}(\mathsf{val}_I(F), \mathsf{val}_I(G)).\\
\mathsf{val}_I(F \leftarrow_3 G) &\stackrel{\text{def}}{=} \mathsf{T} \text{ if } \mathsf{val}_I(F) \geq \mathsf{val}_I(G);\ \mathsf{F} \text{ otherwise.}\\
\mathsf{val}_I(F \leftrightarrow_3 G) &\stackrel{\text{def}}{=} \mathsf{val}_I((F \leftarrow_3 G) \wedge (G \leftarrow_3 F)).
\end{aligned}
\tag{xix}
$$

The variant of implication $\leftarrow_3$ is called $\mathsf{seq_3}$ in the literature [Got01]. The logic specified by $\mathsf{val}$ is known as $\mathsf{S_3}$ in the so-called *standard sequence* of many valued logics [Res69]. Hence we call a three-valued formula in which the only logic operators are those for which there is a clause in the definition (xix) an $\mathsf{S_3}$*-formula*. (In the original presentation of $\mathsf{S_3}$ [Res69] the 0-ary logic operators $\top$ and $\bot$ are not included, but this is no essential difference, since they could be replaced by formulas $p \leftarrow_3 p$ and $\neg(p \leftarrow_3 p)$, respectively, for some arbitrary atom $p$.) If we

speak just of a *formula* in the sequel, we still refer to a formula of classical propositional logic, extended by the operators for circumscription and projection, as specified in Section 2.2.

We now define functions $\mathsf{bin}_n$, for $n \in \{0,1\}$ that map an $\mathsf{S}_3$-formula to a classical propositional formula over scope $0 \cup 1$ in a way that is "compatible" with $\mathsf{val}$, as made precise by the subsequent Proposition 4, which is easy to verify by induction on formulas:

$$\begin{aligned}
\mathsf{bin}_0(A) &\stackrel{\text{def}}{=} A^0. \\
\mathsf{bin}_1(A) &\stackrel{\text{def}}{=} A^1. \\
\mathsf{bin}_n(\bot) &\stackrel{\text{def}}{=} \bot. \\
\mathsf{bin}_n(\top) &\stackrel{\text{def}}{=} \top. \\
\mathsf{bin}_n(\neg F) &\stackrel{\text{def}}{=} \neg \mathsf{bin}_{1-n}(F). \\
\mathsf{bin}_n(F \wedge G) &\stackrel{\text{def}}{=} \mathsf{bin}_n(F) \wedge \mathsf{bin}_n(G). \\
\mathsf{bin}_n(F \vee G) &\stackrel{\text{def}}{=} \mathsf{bin}_n(F) \vee \mathsf{bin}_n(G). \\
\mathsf{bin}_n(F \leftarrow_3 G) &\stackrel{\text{def}}{=} (\mathsf{bin}_0(F) \leftarrow \mathsf{bin}_0(G)) \wedge (\mathsf{bin}_1(F) \leftarrow \mathsf{bin}_1(G)). \\
\mathsf{bin}_n(F \leftrightarrow_3 G) &\stackrel{\text{def}}{=} (\mathsf{bin}_0(F) \leftrightarrow \mathsf{bin}_0(G)) \wedge (\mathsf{bin}_1(F) \leftrightarrow \mathsf{bin}_1(G)).
\end{aligned} \tag{xx}$$

**Proposition 4 (Relating Conversion and Valuation).** *The following properties relate* $\mathsf{bin}_0$ *and* $\mathsf{bin}_1$ *with* $\mathsf{val}$*, for interpretations* $I$ *that are over the predicate groups* 0 *and* 1 *and satisfy* $\mathsf{cons}$*, and* $\mathsf{S}_3$*-formulas* $F$*.*

1. $I \models \mathsf{bin}_0(F)$ *iff* $I \models \mathsf{bin}_0(F) \wedge \mathsf{bin}_1(F)$ *iff* $\mathsf{val}_I(F) = \mathsf{T}$.
2. $I \models \mathsf{bin}_1(F)$ *iff* $\mathsf{val}_I(F) = \mathsf{U}$ *or* $\mathsf{val}_I(F) = \mathsf{T}$.

### 6.2 $\mathsf{S}_3$ and the Reconstructed Fitting Operator Semantics

The following proposition establishes the correspondence between the logic programming semantics obtained by forming Clark's completion and considering it as an $\mathsf{S}_3$-formula on the one hand, and the logic operator $\mathsf{fitting}$ defined in Section 3.4 on the other hand:

**Theorem 3 (Correspondence of Completion under $\mathsf{S}_3$ and the Reconstructed Fitting Operator Semantics).** *Let* $F$ *be the classical representation of a normal logic program and let* $F'$ *be the Clark's completion of the program considered as* $\mathsf{S}_3$*-formula (that is, without assignment of predicate groups and with the logic operators of* $\mathsf{S}_3$ *in place of the corresponding operators of propositional logic). If* $I$ *is an interpretation over predicates in groups* 0 *and* 1*, then the following statements are equivalent:*

1. $I \models \mathsf{least}(\mathsf{fitting}(F, \{\}))$.
2. $I \models \mathsf{least}(\mathsf{cons} \wedge \mathsf{bin}_o(F'))$.
3. $I$ *is a least model of* $F'$.

As shown in [Fit85], the least model of the Fitting operator applied to a normal logic program is the least model of its Clark's completion considered as a three-valued $\mathsf{S}_3$-formula. Hence, from the equivalence of (1.) to (3.) it follows that the logic operator $\mathsf{fitting}$ defined in Section 3.4 correctly renders the semantics of the

Fitting operator with respect to least models. The equivalence of (2.) to (3.), which easily follows from Proposition 4, shows a way to map the three-valued logic corresponding to the Fitting operator to two-valued logic with predicate groups.

### 6.3 Discussion: Alternate Three-Valued Implications

Logic programming semantics have been investigated with respect to different three-valued versions of implication and biconditional. Let us extend the definition of $\mathsf{val}$ in (xix) with definition two further variants of the implication:

$$\begin{aligned}
\mathsf{val}_I(F \leftarrow_1 G) &\stackrel{\text{def}}{=} \mathsf{val}_I(F \leftarrow_3 G) \text{ except for the following cases:} \\
&\quad \mathsf{U} \text{ if } \mathsf{val}_I(F) = \mathsf{U} \text{ and } \mathsf{val}_I(G) = \mathsf{T}; \\
&\quad \mathsf{U} \text{ if } \mathsf{val}_I(F) = \mathsf{F} \text{ and } \mathsf{val}_I(G) = \mathsf{U}. \\
\mathsf{val}_I(F \leftarrow' G) &\stackrel{\text{def}}{=} \mathsf{val}_I(F \leftarrow_1 G) \text{ except for the following case:} \\
&\quad \mathsf{U} \text{ if } \mathsf{val}_I(F) = \mathsf{U} \text{ and } \mathsf{val}_I(G) = \mathsf{U}.
\end{aligned} \tag{xxi}$$

The variants $\leftarrow_1$ and $\leftarrow'$ are termed $\mathsf{seq}_1$ and $\mathsf{seq}'$, respectively, in [Got01]. The variant $\mathsf{seq}_1$ is also called Łukasiewicz's implication, and $\mathsf{seq}'$ is also known as Kleene's strong implication. In the original specification of the Fitting operator semantics [Fit85], for the biconditional the semantics of $\leftrightarrow_3$ has been used. Since only *completed* logic programs were considered as formulas of a three-valued logic, the choice of a three-valued semantics for implication had not been of concern there. It became relevant in the context of other logic programming semantics, and $\leftarrow_3$ has been added as semantics for implication in [Prz89]. Also in the context of human reasoning, the three-valued semantics of implication is of interest. Kleene's strong semantics for implication ($\leftarrow'$) has been erroneously ascribed to a variant of the Fitting operator in [SvL08], which was subsequently corrected in [HR09a] by replacement with Łukasiewicz's implication ($\leftarrow_1$) and the biconditional defined in terms of it as $(F \leftrightarrow_1 G) \stackrel{\text{def}}{=} (F \leftarrow_1 G) \wedge (G \leftarrow_1 F)$. However, for the consideration of a normal logic program under a three-valued semantics, the semantics for implication and biconditional are only relevant as far as, for given argument values, they yield the *designated truth value* $\mathsf{T}$ or not. This is sufficient to decide whether a given interpretation is a model of a given conjunction whose conjuncts are implications or equivalences. In this respect $\leftarrow_3$ and $\leftarrow_1$ are identical. There appears to be no use for the "extra information" produced by $\leftarrow_1$ in yielding $\mathsf{U}$ in some cases where $\leftarrow_3$ yields $\mathsf{F}$, unless logic programs which do allow implications or biconditionals occurring *within* heads or bodies would be considered.

## 7 Conclusion

We summarize the contributions made in this paper, grouped into different aspects, and outline implied further research questions.

### 7.1 Uniform Consideration of Open World Predicates

Logic programming semantics that are applied to model human reasoning following the approach of Stenning and van Lambalgen must allow that only some predicates are handled with closed world reasoning, while others are handled with open world reasoning. Both of these modes are quite naturally available with parallel predicate circumscription, by specifying some predicates to be minimized and others to be fixed. By expressing other logic programming semantics in terms of circumscription, this feature is straightforwardly transferred, yielding variants of these other semantics that allow to specify predicates handled by open world reasoning. Scope-determined circumscription is a variant of circumscription which allows in a first-order context to distinguish between open and closed world reasoning not just on the level of predicates, but also more fine-grained for arbitrary sets of ground atoms. This feature possibly has applications in human reasoning scenarios that are modeled not just propositionally.

### 7.2 Adequacy of Semantics for Modeling Human Reasoning

For the suppression task examples, we have seen that the least models of the partial stable models semantics are exactly the least models obtained with the Fitting operator, when special handling of open world predicates is taken into account. Not only three-valued, but also two-valued semantics, in particular Clark's completion and the stable models semantics, can be considered as modeling the suppression task with no less adequacy, where again open world predicates need to be handled specially.

That completion and stable models semantics yield the same models for the given programs representing the suppression task is no surprise, since these programs satisfy the tightness condition (do not have "positive loops"), also if the predicated for open world reasoning encoded according to Theorem 2. Are there logic programs that represent human reasoning scenarios for which these semantics differ, or do the "simpler" completion semantics suffice generally for human reasoning?

With all logic programming semantics considered here and in the referenced literature, the matching of consequences with those drawn by human subjects in suppression task experiments is rather coarse. Only relative tendencies of human subjects to perform or suppress a conclusion in some of the considered reasoning scenarios compared to certain others are reflected in corresponding conclusions obtained under the logic programming semantics. Considered by themselves, for some of the scenarios, the conclusions obtained under the logic programming semantics do not coincide with those drawn by a significant portion or even majority of the subjects. To obtain a more faithful modeling that yields rationales for different courses of human reasoning, it seems that logic programming semantics have to be considered in further respects than just consequences. This is an open issue, where some possibilities are indicated below in Section 7.5.

### 7.3 More Expressive Logic Programming Languages

For clarity, we based the logic programming semantics on a propositional framework. As shown in [Wer08,Wer10c] this can be straightforwardly generalized to first-order logic, including first-order versions of the logic programming semantics [Wer10a].

The operators like stable that characterize logic programming semantics actually have quite weak preconditions on their arguments: The formula $F$ representing the logic program must be over scope $0 \cup 1 \cup 2$ and the set $O$ of open predicates must satisfy $\mathcal{L}(F) \cap O^0 = \emptyset$. This allows to express various generalizations of normal programs just by passing corresponding formula representations to the operators. For example, generalizations of the stable models semantics by permitting integrity constraints, disjunctive rules and negation as failure in the head can be expressed in this way [Wer10a].

### 7.4 Computational Approaches

A principle that has been followed throughout in the paper is that "semantic" operators are applied as far as possible. That is, logic operators which have classically equivalent values for classically equivalent arguments, independently of syntactic properties. In particular, projection and circumscription are such operators. Logic programming semantics inherit this property by characterizing them in terms of such operators. "Semantic" operators facilitate processing by *general* automated reasoning tools. Projection and circumscription can be processed by variants of second-order quantifier elimination, for which a variety of techniques is available [GSS08,Wer09] and which instantiates with respect to propositional logic to Boolean variable elimination. For any general system, it is a challenge to embed or simulate known efficient techniques for special cases.

A prototype system based on propositional logic has been implemented to explore the approach to the computational processing of logics by eliminating operators such as project and circ (http://cs.christophwernhard.com/provers/toyelim/). A macro feature allows the user to define additional operators like stable in terms of these. Scopes can be specified in a way similar to the notation used in this paper. The system provides a uniform user interface that integrates a portfolio of embedded methods and external programs. input formulas are rewritten such that suitable subproblems can be passed to external QBF or SAT solvers, or be handled by dedicated elimination procedures, which currently are implemented naively, adequate for small applications.

### 7.5 Possible Implications for the Investigation of Human Reasoning

Let us first note what has been suppressed with the approach followed in the paper: A constructive operational view of rule processing, as for example with a fixed point operator on which processing by neural networks can be based [SvL08,HR09b]. Modern logic programming semantics, like stable models, can not be naively characterized in such a constructive way. On the other hand,

ascribing an operational model of the way in which humans process rules remains important with respect to human reasoning. Thus, the mapping of rule semantics to processing models can be identified as a general research issue with respect to human reasoning.

We proceed by pointing out particular aspects of the approach followed in the paper that might be of interest in human reasoning. In the characterizations of logic programming semantics, predicate groups were used to indicate how a particular predicate occurrence has to be handled in two respects: whether it is subjected to minimization by circumscription, and what it contributes to a three-valued valuation ($p^0$ for $p$ is true, $\neg p^1$ for $p$ is false, $\neg p^0$ for $p$ is false or undefined, and $p^1$ for $p$ is true or undefined). It seems to be interesting future research to explore these groups not just considered as a technical encoding device but as associated with epistemic meanings such as *believe*$(p)$ and *potentially-true*$(p)$. Is it useful to take such epistemic annotations into account in the theory of mental models?

We considered logic programs as classical formulas with predicate groups. The assignment of predicate groups is determined by the position of predicate occurrences with respect to the rules of the original program (e.g. an occurrence of $p$ in a head corresponds to $p^0$, an occurrence in a negated body literal to $p^1$). In consequence, logic programming is viewed just as a generalization of circumscription where only some occurrences of a predicate are subjected to minimization. The non-classical rule syntax of logic programs is then just a device to indicate these occurrences. Can this observation be matched with rationales for human reasoning, ways in which humans understand conditionals? Can the "fallacies" observed in human reasoning be explained and systematized as patterns of the interplay of predicate groups that indicate epistemic status?

## References

[Byr89] R. M. J. Byrne. Suppressing valid inferences with conditionals. *Cognition*, 31:61–83, 1989.

[DSSd00] K. Dieussaert, W. Schaeken, W. Schroyen, and G. d'Ydevalle. Strategies during complex conditional inferences. *Thinking and Reasoning*, 6(2):152–161, 2000.

[Fit85] Melvin Fitting. A Kripke-Kleene semantics for logic programs. *Journal of Logic Programming*, 2(4):295–312, 1985.

[Got01] Siegfried Gottwald. *A Treatise on Many-Valued Logics*, volume 9 of *Studies in Logic and Computation*. Research Studies Press, Baldock, UK, 2001.

[GSS08] D. M. Gabbay, R. A. Schmidt, and A. Szałas. *Second-Order Quantifier Elimination: Foundations, Computational Aspects and Applications*. College Publications, London, 2008.

[HPW11] Steffen Hölldobler, Tobias Philipp, and Christoph Wernhard. An abductive model for human reasoning (poster paper). In *Logical Formalizations of Commonsense Reasoning, Papers from the AAAI 2011 Spring Symposium*, AAAI Spring Symposium Series Technical Reports, pages 135–138. AAAI Press, 2011.

[HR09a] S. Hölldobler and C. D. P. Kencana Ramli. Logic programs under three-valued Łukasiewicz's semantics. In *ICLP 2009*, pages 464–478, 2009.

[HR09b] S. Hölldobler and C. D. P. Kencana Ramli. Logics and networks for human reasoning. In *ICANN'09*, pages 85–94, 2009.

[IS98] Katsumi Inoue and Chiaki Sakama. Negation as failure in the head. *Journal of Logic Programming*, 35(1):39–78, 1998.

[JNS+06] Tomi Janhunen, Ilkka Niemelä, Dietmar Seipel, Patrik Simons, and Jia-Huai You. Unfolding partiality and disjunctions in stable model semantics. *ACM Transactions on Computational Logic*, 7(1):1–37, 2006.

[Lif94] Vladimir Lifschitz. Circumscription. In *Handbook of Logic in AI and Logic Programming*, volume 3, pages 298–352. Oxford University Press, Oxford, 1994.

[Lif08] Vladimir Lifschitz. Twelve definitions of a stable model. In *Logic Programming: 24th International Conference, ICLP 2008*, volume 5366 of *LNCS*, pages 37–51. Springer, 2008.

[Lin91] Fangzhen Lin. *A Study of Nonmonotonic Reasoning*. PhD thesis, Stanford University, 1991.

[LLM03] Jérôme Lang, Paolo Liberatore, and Pierre Marquis. Propositional independence – formula-variable independence and forgetting. *JAIR*, 18:391–443, 2003.

[Prz89] Teodor Przymusinski. Every logic program has a natural stratification and an iterated fixed point model. In *Proc. of the 8th Symposium on Principles of Database Systems*, pages 11–21. ACM SIGACT-SIGMOD, 1989.

[Prz90a] Teodor Przymusinski. Extended stable semantics for normal and disjunctive logic programs. In *Proc. of the 7th International Conference on Logic Programming*, pages 459–477. MIT Press, 1990.

[Prz90b] Teodor Przymusinski. Well-founded semantics coincides with three-valued stable semantics. *Fundamenta Informaticae*, 13(4):445–464, 1990.

[Res69] Nicholas Rescher. *Many-Valued Logic*. McGraw-Hill, New York, 1969.

[Sch95] John S. Schlipf. The expressive powers of the logic programming semantics. *Journal of Computer and System Sciences*, 51(1):64–86, 1995.

[SvL05] K. Stenning and M. van Lambalgen. Semantic interpretation as computation in nonmonotonic logic: The real meaning of the suppression task. *Cognitive Science*, (29):916–960, 2005.

[SvL08] Keith Stenning and Michiel van Lambalgen. *Human Reasoning and Cognitive Science*. MIT Press, Cambridge, MA, 2008.

[Wer08] Christoph Wernhard. Literal projection for first-order logic. In *Logics in Artificial Intelligence: 11th European Conference, JELIA 08*, volume 5293 of *LNAI*, pages 389–402. Springer, 2008.

[Wer09] Christoph Wernhard. Tableaux for projection computation and knowledge compilation. In *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference, TABLEAUX 2009*, volume 5607 of *LNAI*, pages 325–340. Springer, 2009.

[Wer10a] Christoph Wernhard. Circumscription and projection as primitives of logic programming. In *Technical Communications of the 26th International Conference on Logic Programming, ICLP'10*, volume 7 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 202–211, 2010.

[Wer10b] Christoph Wernhard. Circumscription and projection as primitives of logic programming – extended version. Technical report, Technische Universität Dresden, 2010. Available from http://cs.christophwernhard.com/papers/logprog2010extended.pdf.

[Wer10c] Christoph Wernhard. Literal projection and circumscription. In *Proceedings of the 7th International Workshop on First-Order Theorem Proving, FTP'09*, volume 556 of *CEUR Workshop Proceedings*, pages 60–74. CEUR-WS.org, 2010.

# A Proofs

## A.1 Notation in Proofs

In proofs we basically use the notation and symbols as specified for the main part of the paper in Section 2. In addition we use the following:

**Notation for Systematic Replacement.** If $i, j, k, l$ are predicate groups, we write a formula $G$ that is over two different predicate groups $i$ and $j$ also as $G[i, j]$ and let $G[k, l]$ denote $G[i, j]$ with all occurrences of predicates from group $i$ replaced by their correspondents from group $k$, and predicates from group $j$ by their correspondents from $l$.

**Notation for Modified Interpretations.** The notation $I[L]$ specifies an interpretation that is like a given one $I$ except that it contains a given literal $L$, no matter whether $I$ contains that literal or not:

$$I[L] \stackrel{\text{def}}{=} (I - \{\overline{L}\}) \cup \{L\}. \tag{xxii}$$

This notation is generalized for sets of literals $M$ that are consistent, that is do not contain a literal and its complement:

$$I[M] \stackrel{\text{def}}{=} (I - \overline{M}) \cup M. \tag{xxiii}$$

**Abbreviations.** In justifications of proof steps we use the following abbreviations:

con. for *contracting the definition of*
def. for *the definition of*
exp. for *expanding the definition of*

## A.2 Proofs of Theorems in Section 5

**Theorem 1 (Encoding Open Predicates with Completion)** *Let $F$ be the classical representation of a normal program and let $O$ be a set of ungrouped predicates such that $\mathcal{L}(F) \cap O^0 = \emptyset$. Define*

$$E \stackrel{\text{def}}{=} \bigwedge_{p \in O} (p^0 \leftarrow p^2).$$

*It then holds that:*

(i) $\mathsf{comp}(F \wedge E, \{\}) \equiv \mathsf{comp}(F, O)$.

(ii) $\mathsf{fitting}(F \wedge E, \{\}) \equiv \mathsf{fitting}(F, O)$.

*Proof (Sketch).* Since $F$ is the classical representation of a normal logic program and $\mathcal{L}(F) \cap O^0 = \emptyset$, we know that $F$ is equivalent to a formula of a particular syntactic form: Let $C \stackrel{\text{def}}{=} \mathcal{P} - O$. For each $p \in C$ there exists a formula $G_p[2,1]$ such that

$$F \equiv \bigwedge_{p \in C} (p^0 \leftarrow G_p[2,1]). \tag{xxiv}$$

Theorem 3.i is then obtained from the following equivalences, which can be shown with the definition of $\mathsf{comp}$ and Proposition 1: $\mathsf{comp}(F \wedge E, \{\}) \equiv \bigwedge_{p \in C}(p^0 \leftrightarrow G_p[0,0]) \wedge \bigwedge_{p \in O}(p^0 \leftrightarrow p^0) \equiv \bigwedge_{p \in C}(p^0 \leftrightarrow G_p[0,0]) \equiv \mathsf{comp}(F, O)$. Theorem 3.ii is obtained from the following equivalences, which can be shown similarly, by taking in addition Proposition 3 into account: $\mathsf{fitting}(F \wedge E, \{\}) \equiv \mathsf{cons} \wedge \bigwedge_{p \in C}((p^0 \leftarrow G_p[0,1]) \wedge (p^1 \rightarrow G_p[1,0])) \wedge \bigwedge_{p \in O}((p^0 \leftarrow p^0) \wedge (p^1 \rightarrow p^1)) \equiv \mathsf{cons} \wedge \bigwedge_{p \in C}((p^0 \leftarrow G_p[0,1]) \wedge (p^1 \rightarrow G_p[1,0])) \equiv \mathsf{fitting}(F, O)$. □

In the proof of Theorem 2 below we make at several times use of the following property of projection:

**Proposition A1 (Modifying Models Outside the Formula Base).** *If* $F \equiv \mathsf{project}_S(F)$, $I \models F$, *and* $I \cap S \subseteq J$, *then* $J \models F$.

**Theorem 2 (Encoding Open Predicates with Stable Models)** *Let $F$ be the classical representation of a normal program and let $O$ be a set of ungrouped predicates such that $\mathcal{L}(F) \cap O^0 = \emptyset$. Assume that for each $p \in O$ there exists a distinguished ungrouped predicate $not\_p$ that is neither in $O$ nor occurs in $F$. Define $N \stackrel{\text{def}}{=} \{not\_p \mid p \in O\}$. Define*

$$E \stackrel{\text{def}}{=} \bigwedge_{p \in O} ((p^0 \leftarrow \neg not\_p^1) \wedge (not\_p^0 \leftarrow \neg p^1)).$$

*It then holds that:*

(iii) $\mathsf{forget}_{N^0}(\mathsf{stable}(F \wedge E, \{\})) \equiv \mathsf{forget}_{N^0}(\mathsf{stable}(F, O))$.

(iv) $\mathsf{forget}_{N^0 \cup N^1}(\mathsf{pstable}(F \wedge E, \{\})) \equiv \mathsf{forget}_{N^0 \cup N^1}(\mathsf{pstable}(F, O))$.

*Proof.* (3.iii) Define $C \stackrel{\text{def}}{=} \mathcal{P} - (O \cup N)$; $F' \stackrel{\text{def}}{=} \mathsf{ren}_{2\setminus 0}(F)$; $H \stackrel{\text{def}}{=} \bigwedge_{not\_p \in N}(not\_p^1 \leftrightarrow not\_p^0)$; and $0 = 1 \stackrel{\text{def}}{=} \bigwedge_{p \in \mathcal{P}}(p^0 \leftrightarrow p^1)$. It then holds that:

$$F' \equiv \mathsf{project}_{C^0 \cup C^1 \cup O^0 \cup O^1}(F'). \tag{xxv}$$

$$E \equiv \mathsf{project}_{O^0 \cup O^1 \cup N^0 \cup N^1}(E). \tag{xxvi}$$

We now bring the two sides of the theorem into forms which are suitable to show their equivalence. By expanding the definitions of $\mathsf{stable}$ and $\mathsf{ren}$, along with properties of projection and circumscription, for left side the following equivalences can be shown:

$$\mathsf{forget}_{N^0}(\mathsf{stable}(F \wedge E, \{\})) \tag{xxvii}$$

$$\equiv \mathsf{forget}_{N^0}(\mathsf{ren}_{1\setminus 0}(\mathsf{circ}_{+0\cup 1}(F' \wedge E))) \tag{xxviii}$$

$$\equiv \mathsf{forget}_{N^0}(\mathsf{forget}_1(1 = 0 \wedge \mathsf{circ}_{+0\cup 1}(F' \wedge E))) \tag{xxix}$$

$$\equiv \mathsf{forget}_1(1 = 0 \wedge \mathsf{forget}_{N^0 \cup N^1}(H \wedge \mathsf{circ}_{+0\cup 1}(F' \wedge E))). \tag{xxx}$$

And for the right side:

$$\begin{aligned}
& \mathsf{forget}_{N^0}(\mathsf{stable}(F, O)) && \text{(xxxi)}\\
\equiv\ & \mathsf{forget}_{N^0}(\mathsf{ren}_{1\setminus 0}(\mathsf{circ}_{+0\cup 1\cup O^0}(F'))) && \text{(xxxii)}\\
\equiv\ & \mathsf{forget}_{N^0}(\mathsf{forget}_1(1 = 0 \wedge \mathsf{circ}_{+0\cup 1\cup O^0}(F'))) && \text{(xxxiii)}\\
\equiv\ & \mathsf{forget}_1(1 = 0 \wedge \mathsf{circ}_{+C^0\cup 1\cup O^0}(F')). && \text{(xxxiv)}
\end{aligned}$$

The theorem follows from the equivalence of the argument formulas of the forgetting operator in formulas (xxx) and (xxxiv), which we are going to show:

$$1 = 0 \wedge \mathsf{forget}_{N^0\cup N^1}(H \wedge \mathsf{circ}_{+0\cup 1}(F' \wedge E)) \ \equiv\ 1 = 0 \wedge \mathsf{circ}_{+C^0\cup 1\cup O^0}(F'). \quad \text{(xxxv)}$$

The left-to-right direction of equivalence (xxxv) can be shown as follows. Consider the following table. Let $I$ be an interpretation that satisfies the left side of (xxxv), but not the right side. Assumptions (1) and (2) follow from this. We show that they lead to a contradiction (25).

| | | |
|---|---|---|
| (1) | $I \models \mathsf{forget}_{N^0\cup N^1}(H \wedge \mathsf{circ}_{+0\cup j}(F' \wedge E))$. | assumption |
| (2) | $I \not\models \mathsf{circ}_{+C^0\cup 1\cup O^0}(F')$. | assumption |
| (3) | There exists a $J$ such that | |
| (4) | $\quad J \models F'$, | } by (1), exp. forget, project |
| (5) | $\quad J \models E$, | |
| (6) | $\quad J \models \neg\mathsf{raise}_{+0\cup 1}(F' \wedge E)$, | |
| (7) | $\quad J - (N^0 \cup N^1) = I - (N^0 \cup N^1)$. | |
| (8) | $I \models F'$. | by (7), (4), (xxv), Prop. A1 |
| (9) | $I \models \mathsf{raise}_{+C^0\cup 1\cup O^0}(F')$. | by (8), (2), exp. circ |
| (10) | There exists a $K$ such that | |
| (11) | $\quad K \models F'$, | } by (9), Prop. A4 |
| (12) | $\quad K \cap (1 \cup O^0) = I \cap (1 \cup O^0)$, | |
| (13) | $\quad K \cap {+C^0} \subset I \cap {+C^0}$. | |
| (14) | Let $K' = K[J \cap (N^0 \cup N^1)]$. | |
| (15) | $K' \models F'$. | by (14), (11), (4), Prop. A1 |
| (16) | $K' \cap (O^0 \cup N^0) = J \cap (O^0 \cup N^0)$. | by (14), (12), (7) |
| (17) | $K' \cap {+C^0} \subset J \cap {+C^0}$. | by (14), (13), (7) |
| (18) | $K' \cap {+0} \subset J \cap {+0}$. | by (17), (16), (1) |
| (19) | $K' \cap N^1 = J \cap N^1$. | by (14) |
| (20) | $K' \cap (O^1 \cup C^1) = J \cap (O^1 \cup C^1)$. | by (14), (27), (7) |
| (21) | $K' \cap 1 = J \cap 1$. | by (20), (19), (1) |
| (22) | $K' \cap (O^0 \cup O^1 \cup N^0 \cup N^1) = J \cap (O^0 \cup O^1 \cup N^0 \cup N^1)$ | by (21), (31) |
| (23) | $K' \models E$. | by (22), (5), (xxvi), Prop. A1 |
| (24) | $J \models \mathsf{raise}_{+0\cup 1}(F' \wedge E)$. | by (23), (21), (18), (15), Prop. A4 |
| (25) | contradiction. | by (39), (6) |

The right-to-left direction of equivalence (xxxv) can be shown as follows. Consider the following table. Let $I$ be an interpretation that satisfies the right side

of equivalence (xxxv), assumptions (1) and (2). We show that $I$ also satisfies the right side, corresponding to steps (1) and (11).

| | | |
|---|---|---|
| (1) | $I \models 1 = 0.$ | assumption |
| (2) | $I \models \mathsf{circ}_{+C^0 \cup 1 \cup O^0}(F').$ | assumption |
| (3) | $I \models F'$ | by (2), exp. circ |
| (4) | $I \models \neg\mathsf{raise}_{+C^0 \cup 1 \cup O^0}(F')$ | by (2), exp. circ |
| (5) | Let $J \stackrel{\text{def}}{=} I[\{+not\_p^0 \mid -p^1 \in I \cap O^1\} \cup \{-not\_p^0 \mid +p^1 \in I \cap O^1\} \cup \{+not\_p^1 \mid -p^0 \in I \cap O^0\} \cup \{-not\_p^1 \mid +p^0 \in I \cap O^0\}].$ | |
| (6) | $J \models F'.$ | by (5), (3), (xxv), Prop. A1 |
| (7) | $J \models E.$ | by (5) |
| (8) | $J - (N^0 \cup N^1) = I - (N^0 \cup N^1).$ | by (5) |
| (9) | $J \models \neg\mathsf{raise}_{+0 \cup 1}(F' \wedge E).$ | shown below |
| (10) | $J \models H.$ | by (5), (1) |
| (11) | $I \models \mathsf{forget}_{N^0 \cup N^1}(H \wedge \mathsf{circ}_{+0 \cup 1}(F' \wedge E))$ | by (10)–(6), con. circ, def. forget, con. project |

It remains to show step (9). We show that assuming its negation, step (12), leads to a contradiction (34).

| | | |
|---|---|---|
| (12) | $J \models \mathsf{raise}_{+0 \cup 1}(F' \wedge E)$ | assumption |
| (13) | There exists a $K$ such that: | |
| (14) | $K \models F'$, | } by (12), Prop. A4 |
| (15) | $K \models E$, | |
| (16) | $K \cap 1 = J \cap 1$, | |
| (17) | $K \cap +0 \subset J \cap +0.$ | |
| (18) | For all members $+p^0$ of $J \cap +O^0$ it holds that: | |
| (19) | $J \models p^0$, | by (18) |
| (20) | $J \models \neg not\_p^1$, | by (19), (5) |
| (21) | $K \models \neg not\_p^1$, | by (20), (16) |
| (22) | $K \models p^0$, | by (21), (15) |
| (23) | $+p^0 \in K \cap +O.$ | by (22), (18) |
| (24) | $K \cap +O^0 \not\subset J \cap +O^0.$ | by (23), (18) |
| (25) | $K \cap +N^0 \not\subset J \cap +N^0.$ | analogously to (24) |
| (26) | $K \cap +C^0 \subset J \cap +C^0.$ | by (25), (24), (17) |
| (27) | $K \cap +C^0 \subset I \cap +C^0.$ | by (26), (5) |
| (28) | Let $K' = K[I \cap N^1].$ | |
| (29) | $K' \models F'.$ | by (28), (16), (xxv)), Prop. A1 |
| (30) | $K' \cap 1 = I \cap 1.$ | by (28), (16), (8) |
| (31) | $K' \cap O^0 = I \cap O^0.$ | by (28), (24), (17), (8) |
| (32) | $K' \cap +C^0 \subset I \cap +C^0.$ | by (28), (27) |
| (33) | $I \models \mathsf{raise}_{+C^0 \cup 1 \cup O^0}(F').$ | by (32)–(29), Prop. A4 |
| (34) | contradiction. | by (33), (4) |

(3.iv) Can be shown analogously to Theorem 3.iii: Define $C \stackrel{\text{def}}{=} \mathcal{P} - (O \cup N)$; $F' \stackrel{\text{def}}{=} \mathsf{cons} \wedge \mathsf{ren}_{[2\backslash 0,1\backslash 3]}(F) \wedge \mathsf{ren}_{[1\backslash 3,0\backslash 1,2\backslash 1,3\backslash 2]}(F)$; $E' \stackrel{\text{def}}{=} \mathsf{ren}_{[2\backslash 0,1\backslash 3]}(E) \wedge$

$\mathsf{ren}_{[1\backslash 3,0\backslash 1,2\backslash 1,3\backslash 2]}(E)$; $H \stackrel{\text{def}}{=} \bigwedge_{not\_p \in N}((not\_p^2 \leftrightarrow not\_p^0) \wedge (not\_p^3 \leftrightarrow not\_p^1))$. It then holds that:

$$F' \equiv \mathsf{project}_{C^0 \cup C^1 \cup C^2 \cup C^3 \cup O^0 \cup O^1 \cup O^2 \cup O^3}(F'). \tag{xxxvi}$$

$$E' \equiv \bigwedge_{p \in O}((p^0 \leftarrow \neg not\_p^3) \wedge (not\_p^0 \leftarrow \neg p^3) \wedge (p^1 \leftarrow \neg not\_p^2) \wedge (not\_p^1 \leftarrow \neg p^2)). \tag{xxxvii}$$

$$E' \equiv \mathsf{project}_{O^0 \cup O^1 \cup O^2 \cup O^3 \cup N^0 \cup N^1 \cup N^2 \cup N^3}(E'). \tag{xxxviii}$$

As in the proof of Proposition (3.iii), the two sides of the theorem can be brought into forms suitable for showing their equivalence. For the left side:

$$\mathsf{forget}_{N^0 \cup N^1}(\mathsf{pstable}(F \wedge E, \{\})) \tag{xxxix}$$

$$\equiv \mathsf{forget}_{2 \cup 3}(2 = 0 \wedge 3 = 1 \wedge \mathsf{forget}_{N^0 \cup N^1 \cup N^2 \cup N^3}(H \wedge \mathsf{circ}_{+0 \cup +1 \cup 2 \cup 3}(F' \wedge E'))). \tag{xl}$$

And for the right side:

$$\mathsf{forget}_{N^0 \cup N^1}(\mathsf{pstable}(F, O)) \tag{xli}$$

$$\equiv \mathsf{forget}_{2 \cup 3}(2 = 0 \wedge 3 = 1 \wedge \mathsf{circ}_{+C^0 \cup +C^1 \cup 2 \cup 3 \cup O^0 \cup O^1}(F')). \tag{xlii}$$

The theorem follows from the equivalence of the argument formulas of the forgetting operator in formulas (xl) and (xlii), which can be shown like equivalence (xxxv) in the proof of Theorem 3.iii, but with $0 \cup 1$ in place of 0; $2 \cup 3$ in place of 1; $E'$ in place of $E$; and $F'$ as well as $H$ as defined above. □

### A.3 Proofs of Theorem in Section 6

In the proof of Theorem 3 we use some additional material. An operator for so-called *raising* compactly expresses a portion of the definition of circ [Wer10c]. It is defined, like circ and proj in Table 1, with a clause that specifies its semantics:

$$I \models \mathsf{raise}_S(F) \text{ iff}_{\text{def}} \text{ there exists a } J \text{ such that } J \models F \text{ and } J \cap S \subset I \cap S. \tag{xliii}$$

Now scope-determined circumscription, and thus also least can be characterized in terms of raising:

**Proposition A2 (Circumscription and Least in Terms of Raising).**

(i) $\mathsf{circ}_S(F) \equiv F \wedge \neg\mathsf{raise}_S(F)$.

(ii) $\mathsf{least}(F) \equiv F \wedge \neg\mathsf{raise}_{+0 \cup -1}(F)$.

Raising is, like projection, monotonic:

**Proposition A3 (Monotonicity of Raising).** *If* $F_1 \models F_2$, *then* $\mathsf{raise}_S(F_1) \models \mathsf{raise}_S(F_2)$.

The following proposition provides an alternate characterization of raising that refers to the *biscope* and *uniscope* of a scope, two disjoint subsets into which a scope can be partitioned: The biscope contains those members of the scope whose complement is also a member of the scope (thus they are "*bi*-polar" members). The uniscope contains the remaining members of the scope, that is, those whose complement is not also a member of the scope (thus they are "*uni*-polar" members). The following definitions provide formal notation for this:

$$\mathsf{bsc}(S) \stackrel{\text{def}}{=} S \cap \overline{S}. \tag{xliv}$$

$$\mathsf{usc}(S) \stackrel{\text{def}}{=} S - \overline{S}. \tag{xlv}$$

Now raising can be characterized as follows:

**Proposition A4 (Raising in Terms of Biscopes and Uniscopes).**

$$I \models \mathsf{raise}_S(F)$$

*if and only if there exists a $J$ such that*

1. $J \models F$,
2. $J \cap \mathsf{bsc}(S) = I \cap \mathsf{bsc}(S)$, *and*
3. $J \cap \mathsf{usc}(S) \subset I \cap \mathsf{usc}(S)$.

**Theorem 3 (Three-Valued Correspondence for Fitting Semantics)** *Let $F$ be the classical representation of a normal logic program and let $F'$ be the Clark's completion of the program considered as $\mathsf{S}_3$-formula (that is, without assignment of predicate groups and with the logic operators of $\mathsf{S}_3$ in place of the corresponding operators of propositional logic). If $I$ is an interpretation over (at least) predicates in groups 0 and 1, then the following statements are equivalent:*

1. $I \models \mathsf{least}(\mathsf{fitting}(F, \{\}))$.
2. $I \models \mathsf{least}(\mathsf{cons} \wedge \mathsf{bin}_o(F'))$.
3. *$I$ is a least model of the $\mathsf{S}_3$-formula $F'$.*

*Proof.* Equivalence of statements (2.) and (3.) follows from Proposition 4. We now show equivalence of (1.) and (2.). Since $F$ is the classical representation of a normal logic program, for each $p \in \mathcal{P}$ there exists a formula $G_p[2,1]$ with $\top, \bot, \neg, \wedge$ and $\vee$ as only logic operators such that

$$F \equiv \bigwedge_{p \in \mathcal{P}} (p^0 \leftarrow G_p[2,1]), \tag{xlvi}$$

and, moreover, for different predicate groups $i$ and $j$, any atoms from group $i$ are only positive in the $G_p[i,j]$, and atoms from $j$ only negative, thus:

$$\text{For all } p \in \mathcal{P} \text{ it holds that } G_p[i,j] \equiv \mathsf{forget}_{-i \cup +j}(G_p[i,j]). \tag{xlvii}$$

With the definition of $\mathsf{fitting}$ and Propositions 1 and 3, from (xlvi) it follows that

$$\mathsf{fitting}(F, \{\}) \equiv \mathsf{cons} \wedge \bigwedge_{p \in \mathcal{P}} ((p^0 \leftarrow G_p[0,1]) \wedge (p^1 \rightarrow G_p[1,0])). \tag{xlviii}$$

With the definition of $\mathsf{bin}_0$ and $\mathsf{bin}_1$, from (xlvi) it follows that

$$\mathsf{cons} \wedge \mathsf{bin}_0(F') \equiv \mathsf{cons} \wedge \bigwedge_{p \in \mathcal{P}} ((p^0 \leftrightarrow G_p[0,1]) \wedge (p^1 \leftrightarrow G_p[1,0])). \tag{xlix}$$

Now let $X, Y$ be formulas defined as follows:

$$X \stackrel{\text{def}}{=} \bigwedge_{p \in \mathcal{P}} ((p^0 \leftarrow G_p[0,1]) \wedge (p^1 \rightarrow G_p[1,0])), \tag{l}$$

$$Y \stackrel{\text{def}}{=} \bigwedge_{p \in \mathcal{P}} ((p^0 \rightarrow G_p[0,1]) \wedge (p^1 \leftarrow G_p[1,0])). \tag{li}$$

Thus, $\mathsf{least}(\mathsf{fitting}(F, \{\})) \equiv \mathsf{least}(\mathsf{cons} \wedge X) \equiv \mathsf{cons} \wedge \mathsf{least}(X)$ and $\mathsf{least}(\mathsf{cons} \wedge \mathsf{bin}_0(F')) \equiv \mathsf{least}(\mathsf{cons} \wedge X \wedge Y) \equiv \mathsf{cons} \wedge \mathsf{least}(X \wedge Y)$. The last step in these equivalences, moving $\mathsf{cons}$ to the outside of $\mathsf{least}$, is justified by Proposition 2. The theorem then follows from the following equivalence, which we are going to show:

$$\mathsf{least}(X) \equiv \mathsf{least}(X \wedge Y). \tag{lii}$$

The left-to-right direction of (lii) can be proven as follows. Consider the table below. The left-to-right direction of Proposition (lii) is derived as step (5), with step (2) remaining to be shown:

| | | |
|---|---|---|
| (1) | $\mathsf{least}(X) \models X$. | by Prop. 4.ii |
| (2) | $\mathsf{least}(X) \models Y$. | shown below |
| (3) | $\neg\mathsf{raise}_{+0\cup-1}(X) \models \neg\mathsf{raise}_{+0\cup-1}(X \wedge Y)$. | by Prop. A3 |
| (4) | $\mathsf{least}(X) \models X \wedge Y \wedge \neg\mathsf{raise}_{+0\cup-1}(X \wedge Y)$. | by (3), (2), (1), Prop. 4.ii |
| (5) | $\mathsf{least}(X) \models \mathsf{least}(X \wedge Y)$. | by (4), Prop. 4.ii |

Step (2) above can be shown as follows. Consider the table below. Let $I$ be an interpretation that satisfies $\mathsf{least}(X)$, step (6). We derive as step (27) that $I$ is also a model of $Y$.

| | | |
|---|---|---|
| (6) | $I \models \mathsf{least}(X)$. | assumption |
| (7) | $I \models X$. | by (6), Prop. 4.ii |
| (8) | $I \models \neg\mathsf{raise}_{+0\cup-1}(X)$. | by (6), Prop. 4.ii |
| (9) | There does not exist a $J$ such that: $J \models X$ and $J \cap (+0 \cup -1) \subset I \cap (+0 \cup -1)$. | by (8) and the def. (xliii) |
| (10) | Let $p$ be an arbitrary member of $\mathcal{P}$. | |
| (11) | Let $X' \stackrel{\text{def}}{=} (p^1 \rightarrow G_p[1,0]) \wedge \bigwedge_{q \in \mathcal{P}-\{p\}}((q^0 \leftarrow G_q[0,1]) \wedge (q^1 \rightarrow G_q[1,0]))$. | |
| (12) | $G_p[0,1] \equiv \mathsf{forget}_{\{-p^0\}}(G_p[0,1])$. | by (xlvii) |
| (13) | $I \models X'$. | by (11), (7), (l) |
| (14) | $X' \equiv \mathsf{forget}_{\{+p^0\}}(X')$. | by (11), (l), (xlvii) |
| (15) | Assume the case that $I \models p^0$. | assumption |
| (16) | $I[-p^0] \not\models X$. | by (15), (9), (7) |

| | | |
|---|---|---|
| (17) | $I[-p^0] \models X'$. | by (14), (13) |
| (18) | $I[-p^0] \not\models p^0 \leftarrow G_p[0,1]$. | by (17), (16), (11), (l) |
| (19) | $I[-p^0] \models G_p[0,1]$. | by (18) |
| (20) | $I \models G_p[0,1]$. | by (19), (12) |
| (21) | $I \models p^0 \rightarrow G_p[0,1]$. | by (20) |
| (22) | Assume complementary to (15) that $I \models \neg p^0$. | assumption |
| (23) | $I \models p^0 \rightarrow G_p[0,1]$. | by (22) |
| (24) | $I \models p^0 \rightarrow G_p[0,1]$. | by (23), (22), (21), (15) |
| (25) | $I \models p^1 \leftarrow G_p[1,0]$. | analogously to (24) |
| (26) | $I \models \bigwedge_{p\in\mathcal{P}}((p^0 \rightarrow G_p[0,1]) \wedge (p^1 \leftarrow G_p[1,0]))$. | by (25), (24), (10) |
| (27) | $I \models Y$. | by (26), (li) |

The right-to-left direction of (lii) can be shown as follows. Consider the following table. We will prove step (1) which is equivalent to step (3), the right-to-left direction of (lii).

| | | |
|---|---|---|
| (1) | $X \wedge Y \wedge \mathsf{raise}_{+0\cup-1}(X) \models \mathsf{raise}_{+0\cup-1}(X \wedge Y)$ | |
| (2) | iff $X \wedge Y \wedge \neg\mathsf{raise}_{+0\cup-1}(X \wedge Y) \models \neg\mathsf{raise}_{+0\cup-1}(X)$ | |
| (3) | iff $\mathsf{least}(X \wedge Y) \models \mathsf{least}(X)$. | by Prop. 4.ii |

Step (1) above can be shown as follows. Consider the following table. Let $I$ be an interpretation that satisfies the left side of (1), that is steps (4)–(6).

| | | |
|---|---|---|
| (4) | $I \models X$. | assumption |
| (5) | $I \models Y$. | assumption |
| (6) | $I \models \mathsf{raise}_{+0\cup-1}(X)$. | assumption |
| (7) | There exists a $J$ such that: | } by (6), (xliii) |
| (8) | $J \models X$ | |
| (9) | $J \cap (+0 \cup -1) \subset I \cap (+0 \cup -1)$. | |
| (10) | There exists a $K$ such that: | } by (7), (5), (4) as shown below |
| (11) | $K \models X$, | |
| (12) | $K \models Y$, | |
| (13) | $K \cap (+0 \cup -1) \subset I \cap (+0 \cup -1)$. | |
| (14) | $I \models \mathsf{raise}_{+0\cup-1}(X \wedge Y)$. | by (10)–(13), (xliii) |

We prove steps (10)–(13) by giving a method for constructing a suitable $K$. The method maintains three parameters $K'$, (the representation of) an interpretation, $Y'$ and $Y''$, conjunctions of implications from $Y$. They are initialized as follows:

$$K' := J;\ Y' := \top;\ Y'' := Y. \tag{liii}$$

The method proceeds in a loop: While there is an implication or reverse implication $C$ in $Y''$ that is not satisfied by $K'$, move $C$ from $Y''$ to $Y'$; if $C$ is of the form $p^0 \rightarrow G_p[0,1]$ for some $p \in \mathcal{P}$, then assign $K' := K'[-p^0]$; otherwise ($C$ must then be of the form $p^1 \leftarrow G_p[1,0]$ for some $p \in \mathcal{P}$) assign $K' := K'[+p^1]$. On exit of the loop, return $K'$ as value of $K$. The loop preserves the following invariants:

(15) $K' \models X \wedge Y'$.
(16) $K' \cap (+0 \cup -1) \subseteq J \cap (+0 \cup -1)$.
(17) $Y' \wedge Y'' \equiv Y$.
(18) For all implications $C$ in $Y'$:
If $C = (p^0 \rightarrow G_p[0,1])$, then $K' \models \neg p^0$; else $C = (p^1 \leftarrow G_p[1,0])$ and $K' \models p^1$.

At termination of the method it holds that $K = K'$ and $K' \models Y''$. The required properties of $K$ then follow with these invariants: (11) from (15); (12) from (15) and (17), since $K \models Y''$; (13) from (16) and (9).

It remains to prove the invariants. That they hold after initialization and that (16)–(18) are preserved by the loop is easy to see. We show (15). Assume that in round $i+1$ of the loop an implication $C = p^0 \rightarrow G_p[0,1]$ is moved from $Y''$ to $Y'$. The case for a reverse implication $p^1 \leftarrow G_p[1,0]$ is analogous. Consider the table below. For natural numbers $n$, let $K'_n$ and $Y'_n$ with denote the values of $K'$ and $Y'$, respectively, when the round $n$ is finished. Since $C$ is moved from $Y''$ to $Y'$ in round $i+1$, it must holds that $K'_i \not\models C$, stated as (19). Assumptions (20) and (21) express the effects the reassignments of $Y'$ and $K'$ performed in round $i+1$. Step (22) and (23) are induction assumptions, corresponding to (15) and (18) above. Let $X'$ be the conjunction of all implications of $X$ with exception of the converse of $C$, that is, $p^0 \leftarrow G_p[0,1]$, as formally expressed in (23). We derive the induction step $K'_{i+1} \models X \wedge Y'_{i+1}$ as step (38):

| | | |
|---|---|---|
| (19) | $K'_i \not\models p^0 \rightarrow G_p[0,1]$. | assumption |
| (20) | $Y'_{i+1} \equiv Y'_i \wedge (p^0 \rightarrow G_p[0,1])$. | assumption |
| (21) | $K'_{i+1} = K'_i[-p^0]$. | assumption |
| (22) | $K'_i \models X \wedge Y'_i$. | assumption |
| (23) | For all implications $D$ in $Y'_i$ there is a $q \in \mathcal{P}$ s. t. either $D = (q^0 \rightarrow G_q[0,1])$ and $K'_i \models \neg q^0$; else $D = (q^1 \leftarrow G_q[1,0])$ and $K'_i \models q^1$. | assumption |
| (24) | Let $X' \stackrel{\text{def}}{=} (p^1 \rightarrow G_p[1,0]) \wedge \bigwedge_{q \in \mathcal{P}-\{p\}}((q^0 \leftarrow G_q[0,1]) \wedge (q^1 \rightarrow G_q[1,0]))$. | |
| (25) | $K'_i \models X$. | by (22) |
| (26) | $X \equiv (p^0 \leftarrow G_p[0,1]) \wedge X'$. | by (24), (l) |
| (27) | $K'_i \models X'$. | by (26), (25) |
| (28) | $K'_i \models p^0 \leftarrow G_p[0,1]$. | by (26), (25) |
| (29) | $K'_i \models \neg G_p[0,1]$. | by (28), (19) |
| (30) | $\neg G_p[0,1] \equiv \mathsf{forget}_{+p^0}(\neg G_p[0,1])$. | by (xlvii) |
| (31) | $K'_i[-p^0] \models p^0 \leftarrow G_p[0,1]$. | by (30) (29) |
| (32) | $X' \equiv \mathsf{forget}_{+p^0}(X')$. | by (24), (xlvii) |
| (33) | $K'_i[-p^0] \models X'$. | by (32), (27) |
| (34) | $K'_i[-p^0] \models X$. | by (33), (31), (26) |
| (35) | $K'_i \models Y'_i$. | by (22) |
| (36) | $K'_i[-p^0] \models Y'_i$. | by (35), (18) |
| (37) | $K'_i[-p^0] \models Y'_{i+1}$. | by (36), (20) |
| (38) | $K'_{i+1} \models X \wedge Y'_{i+1}$. | by (37), (34), (21) |

□