Technical Communications of the International Conference on Logic Programming, 2010 (Edinburgh), pp. 202–211
http://www.floc-conference.org/ICLP-home.html

CIRCUMSCRIPTION AND PROJECTION AS PRIMITIVES OF LOGIC PROGRAMMING

CHRISTOPH WERNHARD

Technische Universität Dresden
E-mail address: christoph.wernhard@tu-dresden.de

Abstract. We pursue a representation of logic programs as classical first-order sentences. Different semantics for logic programs can then be expressed by the way in which they are wrapped into – semantically defined – operators for circumscription and projection. (Projection is a generalization of second-order quantification.) We demonstrate this for the stable model semantics, Clark's completion and a three-valued semantics based on the Fitting operator. To represent the latter, we utilize the polarity sensitiveness of projection, in contrast to second-order quantification, and a variant of circumscription that allows to express predicate minimization in parallel with maximization. In accord with the aim of an integrated view on different logic-based representation techniques, the material is worked out on the basis of first-order logic with a Herbrand semantics.

Introduction

The multitude of semantics for logic programs is traditionally specified by a multitude of techniques: different rule languages, consequence operators, syntactic transformations like *reduct* and *completion*, and notions of model, two- and three-valued, for example. This makes it difficult to uncover relationships and transfer results between the semantics. It lets the long-term goal of a single logic-based system in which a variety of logic programming methods is simultaneously available appear quite fanciful. This work aims towards a unified and integrated view on different semantics for logic programs. We show a framework in which a logic program is represented by a classical first-order sentence, and several semantics for logic programs can be characterized by applying two further logic operators that are *defined in terms of classical semantics*: circumscription and projection.

A key observation is that semantics for logic programs involve circumscription in a way such that only certain *occurrences* of a predicate are affected, while others – basically those in the scope of negation as failure – stay unminimized. Indeed, as shown in [Lin91] and described in [Lif08], the stable models semantics can be characterized accordingly in terms of circumscription. From this point of view, the purpose of a rule syntax is just to indicate which occurrences are to be circumscribed. The alternative pursued here is to replace each "original" predicate symbol by two replicas, one of them used in occurrences where circumscription should take effect. The formula then is classical, permitting for example simplifications that preserve classical equivalence.

Projection, a generalization of second-order quantification, can be used to control the interaction between the replicas. In general, projection is applied in the context of this work to express operations in a semantic way that are typically specified in syntactical terms, like

Technical Communications of the 26th International Conference on Logic Programming, Edinburgh, July, 2010
Editors: Manuel Hermenegildo, Torsten Schaub
LIPIcs - Leibniz International Proceedings in Informatics. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.ICLP.2010.202

systematic renaming of predicate symbols and completion construction, where we refine a semantic characterization in [Lee06].

We apply our framework to the stable model semantics, Clark's completion and a three-valued semantics based on the Fitting operator. The first two are distinguished just by the choice of circumscribed predicate occurrences, reflecting the characterization of Clark's completion in terms of stable models with negation as failure in the head described in [Ino98]. The independence of syntactic constructions lets our framework quite naturally cover extensions of normal logic programs, including disjunctive heads and negation as failure in the head. In accord with the long-term goal of a unified logic-based knowledge processing system, the material in the paper is worked out for first-order logic with a Herbrand semantics, extended by circumscription and projection.

The paper is structured as follows: After notation and the used classical semantics have been specified in Sect. 1, projection and circumscription are introduced in Sect. 2. A view of logic programs as classical first-order sentences is described in Sect. 3. On this basis, it is shown in Sect. 4 how semantics for logic programs are expressed in terms of circumscription and projection. Specifically, the stable model semantics, Clark's completion, and a three-valued semantics based on the Fitting operator are considered. In Sect. 5, the new characterization of the latter is related to the traditional definition, and a similar characterization of partial stable models is sketched. In the conclusion, further potential applications of this framework and a view on computational aspects are indicated.

Please note that this paper is a short version of [Wer10a], which includes additional technical material such as proofs showing correspondence of the discussed and traditional characterizations, and a further notational variant of the described framework that might facilitate its application to prove properties of semantics for logic programs.

1. Notation and Preliminaries

Symbolic Notation. We use the following symbols, also with sub- and superscripts, to stand for items of types as indicated in the following list (precise definitions of the types are given later on), considered implicitly as universally quantified in definition, proposition and theorem statements: F, G, H – Formula; A – Atom; L – Literal; S – Set of ground literals (also called *literal scope*); M – Consistent set of ground literals; I, J, K – Structure; β – Variable assignment. We write the positive (negative) *literal* with atom A as $+A$ ($-A$). The *complement* of literal L is written $\widetilde{L}$. The *set of complements* of a given set S of literals (i.e. $\{\widetilde{L}|L \in S\}$) is written $\widetilde{S}$. We assume a fixed first-order signature with at least one constant symbol. The sets of all ground terms, all ground literals, all positive ground literals, and all negative ground literals – with respect to this signature – are denoted by TERMS, ALL, POS, NEG, respectively. *Variables* are x, y, z, also with subscripts. The sequence $x_1, \ldots, x_n$, where n is the arity of predicate symbol p, is abbreviated by $\overline{x}_p$.

Formulas. We assume that a *formula* is constructed from first-order literals and the logic operators shown in the left column of Tab. 1. That is, we consider formulas of first-order logic, extended by an operator for syntactic equality ($\doteq$) and the two operators project and raise, discussed in Sect. 2. As meta-level notation with respect to this syntax, we use versions of the binary connectives with arbitrary integers ≥ 0 as arity, sequences of

Table 1: The Satisfaction Relation

$\langle I,\beta\rangle \models L$	iff_{def}	$L\beta \in I$
$\langle I,\beta\rangle \models \top$		
$\langle I,\beta\rangle \not\models \bot$		
$\langle I,\beta\rangle \models \neg F$	iff_{def}	$\langle I,\beta\rangle \not\models F$
$\langle I,\beta\rangle \models F_1 \wedge F_2$	iff_{def}	$\langle I,\beta\rangle \models F_1$ and $\langle I,\beta\rangle \models F_2$
$\langle I,\beta\rangle \models F_1 \vee F_2$	iff_{def}	$\langle I,\beta\rangle \models F_1$ or $\langle I,\beta\rangle \models F_2$
$\langle I,\beta\rangle \models \forall x\ F$	iff_{def}	for all $t \in \mathsf{TERMS}$ it holds that $\langle I,\beta\frac{t}{x}\rangle \models F$
$\langle I,\beta\rangle \models \exists x\ F$	iff_{def}	there exists a $t \in \mathsf{TERMS}$ such that $\langle I,\beta\frac{t}{x}\rangle \models F$
$\langle I,\beta\rangle \models t_1 \doteq t_2$	iff_{def}	$t_1\beta = t_2\beta$
$\langle I,\beta\rangle \models \mathsf{project}_S(F)$	iff_{def}	there exists a J such that $\langle J,\beta\rangle \models F$ and $J \cap S \subseteq I$
$\langle I,\beta\rangle \models \mathsf{raise}_S(F)$	iff_{def}	there exists a J such that $\langle J,\beta\rangle \models F$ and $J \cap S \subset I \cap S$

variables as quantifier arguments, and omitting of universal quantifiers. A *sentence* is a formula without free variables. A *clausal sentence* is a sentence $\forall x_1 \dots x_n\ F$, where F is a conjunction with arbitrary arity of disjunctions (*clauses*) with arbitrary arity of literals.

Classical Semantics. We use a notational variant of the framework of Herbrand interpretations: An *interpretation* is a pair $\langle I,\beta\rangle$, where I is a *structure*, that is, a set of ground literals that contains for all ground atoms A exactly one of $+A$ or $-A$, and β is a *variable assignment*, that is, a mapping of the set of variables into TERMS. Formula F with all free variables replaced by their image in β is denoted by $F\beta$; the variable assignment that maps x to ground term t and all other variables to the same values as β is denoted by $\beta\frac{t}{x}$. As explicated in [Wer08], the structure component I of an interpretation $\langle I,\beta\rangle$ represents a *structure* in the conventional sense used in model theory, and, moreover, an interpretation represents a *second-order interpretation* [Ebb84], if predicate variables are considered as distinguished predicate symbols. The satisfaction relation between interpretations and formulas is defined by the clauses in Tab. 1. Entailment and equivalence are straightforwardly defined in terms of it. Entailment: $F_1 \models F_2$ holds if and only if for all $\langle I,\beta\rangle$ such that $\langle I,\beta\rangle \models F_1$ it holds that $\langle I,\beta\rangle \models F_2$. Equivalence: $F_1 \equiv F_2$ if and only if $F_1 \models F_2$ and $F_2 \models F_1$.

2. Projection, Literal Scopes and Circumscription

The project operator, defined semantically in Tab. 1, is applied in the context of this paper to provide *semantic* characterizations of operations and properties that are typically defined in syntactic terms: Clark's completion, extracting the subformula with the "converse rules" from Clark's completion, systematic renaming of predicate symbols, and independence of a formula from given predicate symbols. The formula $\mathsf{project}_S(F)$ is called the *projection* of formula F onto literal scope S. The *forgetting* in F about S is a variant of projection, where the scope is considered complementary:

Definition 1 (Forgetting)**.** $\mathsf{forget}_S(F) \stackrel{\text{def}}{=} \mathsf{project}_{\mathsf{ALL}-S}(F)$.

We call a set of ground literals in the role as argument to projection a *literal scope*. When specifying literal scopes, we let a set of predicate symbols stand for the set of all ground instances of literals whose predicate symbol is in the set.

As an intuitive special case of projection, consider a literal scope S that contains the same atoms in positive as well as negative literals. The condition $J \cap S \subseteq I$ in the definition of $\mathsf{project}$ is then equivalent to $J \cap S = I \cap S$, that is, structures I and J are required to be equal as far as members of S are considered, but unrelated otherwise. Projection is a generalization of second-order quantification: if S is the set of all ground literals with a predicate symbol other than p, then $\mathsf{project}_S(F)$ (or equivalently $\mathsf{forget}_{\{\mathsf{p}\}}(F)$) can be expressed by the second-order formula $\exists \mathsf{p}\, F$.

Beyond second-order quantification, the condition $J \cap S \subseteq I$ in the definition of $\mathsf{project}$ encodes a different effect on literals depending on whether they are positive or negative (w.r.t. to formulas that do not contain $\neg$). Hence, this variant of projection is also termed *literal projection*. Consider for example, $\mathsf{forget}_{\{+\mathsf{q},-\mathsf{q}\}}((+\mathsf{p} \vee -\mathsf{q}) \wedge (+\mathsf{q} \vee -\mathsf{r}))$ which is equivalent to $(+\mathsf{p} \vee -\mathsf{r})$, and, in contrast, $\mathsf{forget}_{\{+\mathsf{q}\}}((+\mathsf{p} \vee -\mathsf{q}) \wedge (+\mathsf{q} \vee -\mathsf{r}))$ which is equivalent to $((+\mathsf{p} \vee -\mathsf{q}) \wedge (+\mathsf{p} \vee -\mathsf{r}))$, where $-\mathsf{q}$ is retained. In the context of this paper, these effects are applied to specify a three-valued semantics for logic programs. Further material on projection can be found in [Wer08]. The other "nonstandard" operator defined in Tab. 1 is raise, which we apply to define *scope-determined circumscription* [Wer10b], a generalization of predicate circumscription [McC80]:

Definition 2 (Scope-Determined Circumscription)**.** $\mathsf{circ}_S(F) \stackrel{\text{def}}{=} F \wedge \neg \mathsf{raise}_S(F)$.

The argument S is also a literal scope, which then provides a uniform interface for expressions combining projection and circumscription. Superficially, raise is very similar to $\mathsf{project}$: Consider Tab. 1. The definition of $\mathsf{project}$ is equivalent to $J \cap S \subseteq I \cap S$. Just by replacing the subset relation ($\subseteq$) with strict subset ($\subset$), the definition of raise is obtained. If F is a sentence over disjoint sets of predicate symbols P, Q and Z, then the *parallel predicate circumscription of P in F with fixed Q and varied Z* [Lif94], traditionally written $\mathrm{CIRC}[F; P; Z]$, is expressed as $\mathsf{circ}_{(P \cap \mathsf{POS}) \cup Q}(F)$. Recall that in specifications of literal scopes, we let a set of predicate symbols stand for the set of all ground instances of literals whose predicate symbol is in the set. The scope $(P \cap \mathsf{POS}) \cup Q$ thus is the set of all *positive* ground literals with a circumscribed predicate symbol, and *all* ground literals with a fixed predicate symbol. While circumscription traditionally just allows to express predicate *minimization*, scope-determined circumscription symmetrically permits to express *maximization* by scopes containing just *negative* ground literals with predicate symbols to be maximized. In the context of this paper, parallel minimization and maximization is applied to specify a three-valued semantics for logic programs.

3. Logic Programs as Classical Sentences

A logic program is typically understood as a set of rules of the form:

$$A_1 \mid \ldots \mid A_k \mid \mathsf{not}\, A_{k+1} \mid \ldots \mid \mathsf{not}\, A_l \leftarrow A_{l+1}, \ldots, A_m, \mathsf{not}\, A_{m+1}, \ldots, \mathsf{not}\, A_n. \tag{3.1}$$

This involves logic operators which do not belong to classical first-order logic. To represent a logic program as a classical first-order sentence, we assume that the set of all predicate symbols can be partitioned into *predicate groups*, that is, disjoint sets of equal cardinality. The idea is that each "original predicate symbol" is replicated once in each group. The respective copy of the "original symbol" p in predicate group P is then written p^P. If P and Q are two predicate groups, we say that p^P and p^Q are *corresponding* predicate symbols, assuming that they have the same arity, which we also call *arity of p*. We transfer

the notation p^P to atoms and literals: A^P (L^P) stands for an atom (literal) whose predicate symbol is in predicate group P. Formally, the partitioning into predicate groups can be modeled by means of a total ordering $<_{\mathsf{pred}}$ on predicate symbols such that p denotes the position of p^P within predicate group P sorted according to $<_{\mathsf{pred}}$. Corresponding predicate symbols then have the same positions within their respective group. The set of all such positions p is written PREDS.

Definition 3 (Predicate Groups $\mathcal{C}, \mathcal{F}, \mathcal{O}$)**.** The symbols $\mathcal{C}, \mathcal{F}, \mathcal{O}$ denote three different predicate groups.

Predicate groups $\mathcal{C}, \mathcal{F}, \mathcal{O}$ are used to express logic programs. Roughly, the group indicates whether a predicate occurrence should be *circumscribed* (group $\mathcal{C}$), should be *fixed* with respect to circumscription (group $\mathcal{F}$), or is yet *open* (group $\mathcal{O}$), that is, further operations are applied that place it into group $\mathcal{C}$ or $\mathcal{F}$ at a later stage.

Definition 4 (Rule Clause, Raw Rule Clause)**.** (i) A *rule clause* is a clause of the form

$$+A_1^{\mathcal{C}} \vee \ldots \vee +A_k^{\mathcal{C}} \vee -A_{k+1}^{\mathcal{F}} \vee \ldots \vee -A_l^{\mathcal{F}} \vee -A_{l+1}^{\mathcal{C}} \vee \ldots \vee -A_m^{\mathcal{C}} \vee +A_{m+1}^{\mathcal{F}} \vee \ldots \vee +A_n^{\mathcal{F}},$$

where $n \geq m \geq l \geq k \geq 0$.
(ii) A *raw rule clause* is like a rule clause, except that the literals with indexes from $l+1$ to m are from predicate group $\mathcal{O}$ instead of $\mathcal{C}$.

Based on Def. 4, a logic program can be understood as a clausal sentence with rule clauses or raw rule clauses. In both cases, a logic program is then just a *classical first-order sentence* that meets certain restrictions. ([Raw] rule clauses can contain universal variables.) When we say that a [raw] rule clause *corresponds* to a rule of the form (3.1), we assume that the [raw] rule clause has predicate symbols from groups as indicated by matching (3.1) with Def. 4.

The *head* of a [raw] rule clause is the disjunction of those of its literals whose index is less or equal to l, its *body* is the conjunction of the complements of its literals with index greater than l. A [raw] rule clause can express a *normal rule* (if $k = l = 1$), *integrity constraint* (if $k = l = 0$), *disjunctive rule* (if $k = l > 1$) and a *rule with negation as failure in the head* (if $l > k$). The class of rules in *general extended disjunctive programs (GEDP)* considered in [Ino98] is however strictly more general: In rules of the form (3.1), GEDP would allow also *negated* atoms in place of the atoms A_i, for $i \in \{1, \ldots, n\}$.

Predicate Renaming. Definition 6 below gives a semantic account of systematically replacing predicate symbols from one group P by their correspondents from another group Q. First we define of shorthands for formulas that will be used at several places in the sequel.

Definition 5 (Predicate Inclusion)**.** Let P, Q be predicate groups. (i) $P \leq Q \stackrel{\text{def}}{=} \forall \overline{x} \bigwedge_{p \in \mathsf{PREDS}} (-p^P(\overline{x}_p) \vee +p^Q(\overline{x}_p))$; (ii) $P = Q \stackrel{\text{def}}{=} (P \leq Q) \wedge (Q \leq P)$; (iii) $P < Q \stackrel{\text{def}}{=} (P \leq Q) \wedge \neg(Q \leq P)$.

Definition 6 (Predicate Renaming in Terms of Projection)**.** Let P, Q, P be predicate groups. Then $\mathsf{rename}_{P \setminus Q}(F) \stackrel{\text{def}}{=} \mathsf{forget}_P(F \wedge P{=}Q)$. Notation $\mathsf{rename}_{[P_1 \setminus P_2, \ldots, P_{n-1} \setminus P_n]}(F)$ is a shorthand for $\mathsf{rename}_{P_{n-1} \setminus P_n}(\ldots(\mathsf{rename}_{P_1 \setminus P_2}(F))\ldots)$.

The formula $\mathsf{rename}_{P \setminus Q}(F)$ is equivalent to F with all occurrences of predicate symbols from P replaced by their respective corresponding predicate symbols from Q.

4. Semantics for Logic Programs via Circumscription and Projection

Based on the representation of a logic program as a clausal first-order sentence with raw rule clauses, three well-known semantics for logic programs – the stable model semantics, the classical models of Clark's completion, and the three-valued minimal models obtained with the Fitting operator – can be characterized in terms of circumscription and projection:

Definition 7 (Semantics For Logic Programs)**.** Let F be a formula over $\mathcal{C} \cup \mathcal{F} \cup \mathcal{O}$.

(i) $\mathsf{ans\text{-}stable}(F) \stackrel{\text{def}}{=} \mathsf{rename}_{\mathcal{F}\backslash\mathcal{C}}(\mathsf{circ}_{(\mathcal{C}\cap\mathsf{POS})\cup\mathcal{F}}(\mathsf{rename}_{\mathcal{O}\backslash\mathcal{C}}(F)))$.

(ii) $\mathsf{ans\text{-}completion}(F) \stackrel{\text{def}}{=} \mathsf{rename}_{\mathcal{F}\backslash\mathcal{C}}(\mathsf{circ}_{(\mathcal{C}\cap\mathsf{POS})\cup\mathcal{F}}(\mathsf{rename}_{\mathcal{O}\backslash\mathcal{F}}(F)))$.

(iii) $\mathsf{ans\text{-}fitting}(F) \stackrel{\text{def}}{=} \mathsf{circ}_{(\mathcal{C}\cap\mathsf{POS})\cup(\mathcal{F}\cap\mathsf{NEG})}(\mathcal{C} \leq \mathcal{F} \wedge \mathsf{rename}_{\mathcal{O}\backslash\mathcal{C}}(F) \wedge F^*)$,

where $F^* = \mathsf{rename}_{[\mathcal{C}\backslash\mathcal{O},\, \mathcal{F}\backslash\mathcal{C},\, \mathcal{O}\backslash\mathcal{F}]}(\mathsf{forget}_{\mathcal{C}\cap\mathsf{POS}}(\mathsf{circ}_{(\mathcal{C}\cap\mathsf{POS})\cup\mathcal{O}\cup\mathcal{F}}(F)))$.

The definientia are formulas of first-order logic extended with project (recall that rename is a shorthand for a formula with project) and circ as additional operators. For ans-stable and ans-completion, the involved projection could also be expressed as second-order quantification, as indicated in Sect. 2, and the involved scope-determined circumscription corresponds to parallel predicate circumscription of $\mathcal{C}$ with fixed $\mathcal{F}$. For ans-fitting, in contrast, proper generalizations of second-order quantification and parallel predicate circumscription are utilized: The scope $\mathcal{C} \cap \mathsf{POS}$ of the forgetting is just about *positive literals* with a predicate from $\mathcal{C}$. The scope $(\mathcal{C}\cap\mathsf{POS})\cup(\mathcal{F}\cap\mathsf{NEG})$ of the outer circumscription expresses minimization of $\mathcal{C}$ in parallel with *maximization* of $\mathcal{F}$.

Semantics for logic programs are usually specified in terms of sets of atoms (*answer sets*), or "partial interpretations", that is, consistent sets of literals, representing a three-valued assignment of atoms to truth values: *true* (*false*) for the atoms of positive (negative) literals in the set, and *undefined* for the remaining atoms. In contrast, semantics for logic programs are specified in Def. 7 as classical models. For ans-stable, such a classical model $\langle I, \beta\rangle$ corresponds to the answer set $\{A \mid +A^{\mathcal{C}} \in I\}$. Predicates from $\mathcal{F}$ are not considered for the answer set, reflecting that $\mathcal{F}$ is forgotten by the outer rename. For ans-fitting, $\langle I, \beta\rangle$ corresponds to the partial interpretation $\{+A \mid +A^{\mathcal{C}} \in I\} \cup \{-A | -A^{\mathcal{F}} \in I\}$.

The characterization of stable models in terms of circumscription (Def. 7.i) originates from [Lin91] and is described as *"definition F"* in [Lif08] for logic programs over $\mathcal{C} \cup \mathcal{F}$ (in our notation). We use the third group $\mathcal{O}$ for mapping to other semantics. In [Fer07] a characterization of stable models in terms of a formula translation that is *similar* to predicate circumscription has been presented. Roughly, it differs from circumscription in that only certain *occurrences* of predicates are circumscribed. In this respect it is like the approach pursued here. However, in [Fer07] these occurrences are identified by their syntactic position within formulas from a fragment of classical propositional logic – to the effect, that classically equivalent programs might not be equivalent when considered as logic programs. For a formal proof of the correspondence of ans-stable to the original characterization of stable models [Gel88] and variants of it see [Wer10a].

Equivalence of ans-completion to the syntactically defined Clark's completion [Cla78] is shown in [Wer10a] along the approach of [Lee06], but generalized to first-order logic and refined by utilizing predicate groups: Head literals are distinguished by placing them in $\mathcal{C}$, which allows to prove *equivalence* of semantic and syntactic characterizations, whereas the related Proposition 4 in [Lee06] just makes the weaker statement that the semantically defined completion of F_1 is equivalent to the syntactically defined Clark's completion of F_2 for *some* F_2 *that is equivalent to* F_1.

The formula F in Def. 7 is over $\mathcal{C} \cup \mathcal{F} \cup \mathcal{O}$. In ans-stable and ans-completion, it is subjected to renaming the predicate symbols from $\mathcal{O}$ to either $\mathcal{C}$ or $\mathcal{F}$, respectively, which is actually the only difference between these semantics. For F that are just over $\mathcal{C} \cup \mathcal{F}$ both semantics are identical. The characterization of Clark's completion in terms of stable models of programs with negation as failure in the head, described by means of a program transformation in [Ino98], thus can be rendered by the following equivalence:

$$\text{If } F \text{ is over } \mathcal{C} \cup \mathcal{F} \cup \mathcal{O}, \text{ then } \mathsf{ans\text{-}completion}(F) \equiv \mathsf{ans\text{-}stable}(\mathsf{rename}_{\mathcal{O} \setminus \mathcal{F}}(F)). \tag{4.1}$$

Based on a fixed-point characterization of the models of Clark's completion as so-called *supported models* [Apt88], it has been shown in [Mar92] that a stable model of a *normal* logic program (i.e. with rules of the form (3.1) where $k = l = 1$) is also a minimal model of its Clark completion. For more general classes of logic programs, analogous properties can be proven on the basis of Def. 7: If F is over $\mathcal{C} \cup \mathcal{F} \cup \mathcal{O}$ and $F \equiv \mathsf{forget}(F, \mathcal{O} \cap \mathsf{POS})$, then $\mathsf{ans\text{-}stable}(F) \models \mathsf{ans\text{-}completion}(F)$ and, if in addition, $F \equiv \mathsf{forget}(F, \mathcal{F} \cap \mathsf{NEG})$, then $\mathsf{ans\text{-}stable}(F) \models \mathsf{circ}_{\mathcal{C} \cap \mathsf{POS}}(\mathsf{ans\text{-}completion}(F))$ [Wer10a].

5. Three-Valued Semantics Based on the Fitting Operator

In [Fit85] a consequence operator Φ (*Fitting operator*) is introduced which is applied to construct three-valued interpretations M, represented by consistent sets of ground literals. For a ground program F with rules of the form (3.1), constrained by $k = l = 1$ (i.e. normal rules), the value of the Fitting operator can be described as follows: The body of a rule is *true* with respect to M, if and only if each of its literals is contained in M. It is *false* with respect to M if and only if the complement of at least one of its literals is in M. For a given M, the Fitting operator yields the union of (1.) the set of all positive literals $+A$ such that there exists a rule of F with head $+A$ whose body is true with respect to M, and (2.) the set of all negative literals $-A$ such all rules of F with head $+A$ have a body that is false with respect to M. The minimal fixed point (minimal w.r.t. set inclusion of the consistent literal sets M) of the Fitting operator then represents a (partial) model, the result of the program, and thus might be called "answer set" according to "Fitting's semantics".

To show that ans-fitting (Def. 7.iii) corresponds to this semantics, we reconstruct it in our framework. We use interpretations over the union of the two predicate groups $\mathcal{C}$ and $\mathcal{F}$ to represent the consistent literal sets expressing three-valued or interpretations. Structures I such that

$$\langle I, \beta \rangle \models \mathcal{C} \leq \mathcal{F} \tag{5.1}$$

(assignment β is irrelevant for (5.1) since $\mathcal{C} \leq \mathcal{F}$ does not contain free variables) are mapped with the following one-to-one correspondence to such literal sets M: $\mathsf{litset}(I) \stackrel{\text{def}}{=} \{+A \mid +A^{\mathcal{C}} \in I\} \cup \{-A \mid -A^{\mathcal{F}} \in I\}$; and $\mathsf{litset}^{-1}(M) \stackrel{\text{def}}{=} \{+A^{\mathcal{C}} \mid +A \in M\} \cup \{-A^{\mathcal{C}} \mid +A \notin M\} \cup \{+A^{\mathcal{F}} \mid -A \notin M\} \cup \{-A^{\mathcal{F}} \mid -A \in M\}$. Minimization with respect to set inclusion of the literal sets M can be expressed by scope-determined circumscription with scope $S = (\mathcal{C} \cap \mathsf{POS}) \cup (\mathcal{F} \cap \mathsf{NEG})$, since $\mathsf{litset}(I) \subseteq \mathsf{litset}(J)$ if and only if $I \cap S \subseteq J$. The scope S effects that predicates from $\mathcal{C}$ are minimized, and, in parallel, predicates from $\mathcal{F}$ are *maximized*, which can not be directly expressed by conventional predicate circumscription.

The Fitting operator is – like the original form of Clark's completion – applied to normal logic programs, that is, sets of rules of the form (3.1) where $k = l = 1$. Such a program corresponds to a clausal sentence with rule clauses that are over $\mathcal{F}$ except possibly for a single positive literal over $\mathcal{C}$. For Clark's completion, in a first "preprocessing" step, such a

sentence is transformed to an equivalent, possibly nonclausal, sentence of a second particular form, which is then the basis for the proper completion transformation. A suitable such second form will be specified in Def. 8 below. We call it *normal completion input sentence*, since any clausal sentence with rule clauses constrained by $k = l = 1$ is equivalent to such a sentence, obtainable by straightforward rewriting with equivalences, including

$$+p(t_1, \ldots, t_n) \vee G \;\equiv\; \forall x_1 \ldots x_n \, +p(x_1, \ldots, x_n) \vee \neg x_1 \doteq t_1 \vee \ldots \vee \neg x_n \doteq t_n \vee G, \qquad (5.2)$$

where $x_1, \ldots, x_n$ are variables not occurring in $t_1, \ldots, t_n, G$.

Definition 8 (Normal Completion Input Sentence)**.** A sentence F is called a *normal completion input sentence* if it is over $\mathcal{C} \cup P$, with P being a set of predicate symbols not in $\mathcal{C}$, and is of the form $\forall \overline{x} \; (\bigwedge_{p \in \mathsf{PREDS}} (+p^{\mathcal{C}}(\overline{x}_p) \vee G_p(\overline{x}_p)))$, where (1.) $\overline{x}$ is $x_1, \ldots, x_k$, with k being the maximal arity of all members of PREDS, (2.) $G_p(\overline{x}_p)$ are formulas whose free variables are in $\overline{x}_p$, (3.) $G_p(\overline{x}_p)$ does not contain predicate symbols from $\mathcal{C}$.

In traditional terminology, a subformula $+p^{\mathcal{C}}(\overline{x}_p)$ of a normal completion input sentence corresponds to a head, and $G_p(\overline{x}_p)$ to the negated disjunction of all bodies of clauses with head $+p^{\mathcal{C}}(\overline{x}_p)$. For a normal completion input sentence F over $\mathcal{C} \cup \mathcal{F}$, Clark's completion of F can then be defined as $\mathsf{rename}_{\mathcal{F} \setminus \mathcal{C}}(F \wedge F^*)$, where F^* is the syntactic completion addendum of F, defined as follows:

Definition 9 (Syntactic Completion Addendum)**.** Let F be a normal completion input sentence with syntactic constituents as specified in Def. 8. The following sentence is called the *syntactic completion addendum* of F: $\forall \overline{x} \; (\bigwedge_{p \in \mathsf{PREDS}} (-p^{\mathcal{C}}(\overline{x}_p) \vee \neg G_p(\overline{x}_p)))$.

Let F be a normal completion input sentence over $\mathcal{C} \cup \mathcal{F} \cup \mathcal{O}$. Recall the definition of ans-fitting (Def. 7.iii):

$$\mathsf{ans\text{-}fitting}(F) \;\stackrel{\text{def}}{=}\; \mathsf{circ}_{(\mathcal{C} \cap \mathsf{POS}) \cup (\mathcal{F} \cap \mathsf{NEG})}(\mathcal{C} \leq \mathcal{F} \wedge \mathsf{rename}_{\mathcal{O} \setminus \mathcal{C}}(F) \wedge F^*),$$

where $F^* = \mathsf{rename}_{[\mathcal{C} \setminus \mathcal{O}, \, \mathcal{F} \setminus \mathcal{C}, \, \mathcal{O} \setminus \mathcal{F}]}(\mathsf{forget}_{\mathcal{C} \cap \mathsf{POS}}(\mathsf{circ}_{(\mathcal{C} \cap \mathsf{POS}) \cup \mathcal{O} \cup \mathcal{F}}(F)))$. The outer circumscription has the scope S discussed above in this section, and thus effects minimization to the smallest models with respect to the three-valued view of interpretations. The argument formula of this circumscription consists of three conjuncts. The first one is (5.1) which excludes interpretations without a consistent three-valued correspondence. The other ones correspond to the positive and negative consequences, respectively, of the Fitting operator.

Assume that the normal completion input sentence F has been obtained in a "preprocessing" step, as outlined above, from an equivalent clausal sentence with raw clauses, representing a conjunction F_0 of rules of the form (3.1), constrained by $k = l = 1$, and such that all heads have just mutually distinct variables as argument terms (in presence of equivalence (5.2) the last condition is w.l.o.g.). Let p be some member of PREDS. The formula $G_p(\overline{x}_p)$ is then a constituent of F as specified in Def. 8. The second conjunct $\mathsf{rename}_{\mathcal{O} \setminus \mathcal{C}}(F)$ is the original logic program, with $\mathcal{O}$ renamed to $\mathcal{C}$, as in ans-stable. It can be shown that $\langle I, \beta \rangle \models \mathcal{C} \leq \mathcal{F} \wedge \mathsf{rename}_{\mathcal{O} \setminus \mathcal{C}}(\neg G_p(\overline{x}_p))$ if and only if there is a rule R with head predicate p in F_0 such that the body of its ground instance $R\beta$ is *true* with respect to $\mathsf{litset}(I)$. The subformulas $(+p^{\mathcal{C}}(\overline{x}_p) \vee \mathsf{rename}_{\mathcal{O} \setminus \mathcal{C}}(\neg G_p(\overline{x}_p)))$ in $\mathsf{rename}_{\mathcal{O} \setminus \mathcal{C}}(F)$ then allow to infer positive literals with predicate $p^{\mathcal{C}}$, corresponding to positive consequences of the Fitting operator.

Analogously, $\langle I, \beta \rangle \models \mathcal{C} \leq \mathcal{F} \wedge \mathsf{rename}_{[\mathcal{F} \setminus \mathcal{C}, \, \mathcal{O} \setminus \mathcal{F}]}(G_p(\overline{x}_p))$ if and only if for all rules R with head predicate p in F_0 it holds that the body of the ground instance $R\beta$ is *false* with

respect to $\mathsf{litset}(I)$. Subformulas of the form $(-p^{\mathcal{F}}(\overline{x}_p) \vee \mathsf{rename}_{[\mathcal{F}\setminus\mathcal{C},\, \mathcal{O}\setminus\mathcal{F}]}(G_p(\overline{x}_p)))$ then allow to infer negative literals with predicate $p^{\mathcal{F}}$, corresponding to negative consequences of the Fitting operator. Sentence F^* is the universally quantified conjunction of these subformulas, for each predicate p from PREDS. It is equivalent to the syntactic completion addendum of F (Def. 9), subjected to switching group assignments $\mathcal{C}$ and $\mathcal{F}$, and renaming $\mathcal{O}$ to $\mathcal{F}$. This switching and renaming is expressed by rename applied to $[\mathcal{C}\setminus\mathcal{O},\ \mathcal{F}\setminus\mathcal{C},\ \mathcal{O}\setminus\mathcal{F}]$. As shown in [Wer10a], the circumscription in F^* is equivalent to Clark's completion of F. The forgetting about $\mathcal{C} \cap \mathsf{POS}$ serves to extract an equivalent to the *syntactic completion addendum* from the completion, according to the following theorem proven in [Wer10a]:

Theorem 5.1 (Semantic Extraction of the Completion Addendum)**.** *Let F be a completion input sentence and F^* its completion. Then $\mathsf{forget}_{\mathcal{C}\cap\mathsf{POS}}(F^*)$ is equivalent to the syntactic completion addendum of F.*

Literal projection is utilized there to preserve the negative literals from $\mathcal{C}$ in the addendum, but forget about the positive literals from $\mathcal{C}$ in the original formula, and with them the whole original formula.

A further prominent semantics for logic programs with three-valued models is the *partial stable model semantics*. In [Jan06] a characterization of partial stable models as stable models of a translated program is given (tracing back to earlier work [Sch95]). Based on a reconstruction of the syntactic transformation Tr(P) of [Jan06] in terms of rename, and on the characterization of stable models by $\mathsf{ans\text{-}stable}$, partial stable models can be characterized in our framework as shown in the following definition. The three-valued (i.e. partial) interpretations are represented there in the same way as shown above for the Fitting semantics.

Definition 10. Let F be a formula over $\mathcal{C} \cup \mathcal{F} \cup \mathcal{O}$. Let $\mathcal{C}'$ and $\mathcal{F}'$ be two additional predicate groups, different from each other and from $\mathcal{C}, \mathcal{F}, \mathcal{O}$.

$$\mathsf{ans\text{-}partial\text{-}stable}(F) \stackrel{\text{def}}{=} \mathsf{rename}_{[\mathcal{C}'\setminus\mathcal{C},\mathcal{F}'\setminus\mathcal{F}]}(\mathsf{circ}_{((\mathcal{C}\cup\mathcal{F})\cap\mathsf{POS})\cup\mathcal{C}'\cup\mathcal{F}'}(\mathcal{C} \leq \mathcal{F} \wedge F_1 \wedge F_2)),$$

where $F_1 = \mathsf{rename}_{[\mathcal{O}\setminus\mathcal{C},\mathcal{F}\setminus\mathcal{F}']}(F)$ and $F_2 = \mathsf{rename}_{[\mathcal{O}\setminus\mathcal{C},\mathcal{F}\setminus\mathcal{C}',\mathcal{C}\setminus\mathcal{F}]}(F)$.

6. Conclusion

We investigated a representation of logic programs as classical first-order sentences that are wrapped into the semantically defined additional operators circumscription and projection, in different ways, rendering different established semantics of logic programs. The generality of our framework indicates interesting spaces that have yet to be explored: Our characterizations of semantics for logic programs apply to broad formula classes. The scopes of circumscription and projection in the characterizations of semantics could be modified, or additional applications of projection could be merged in, to express, for example, models that are "stable only with respect to some atoms", and to restrict answer sets to atoms that are relevant for the user [Eit08, Geb09].

A computational approach to the processing of operators for circumscription and projection is "elimination", analogous to second-order quantifier elimination: Computing for a given formula that involves the operator (second-order quantifier, resp.) an equivalent formula without the operator (second-order quantifier, resp.). Indeed, methods for the computation of circumscription and projection can essentially be considered as methods for

second-order quantifier elimination [Gab08, Wer08, Wer09]. Our framework thus indicates that methods for processing logic programs could be seen in this context: On one hand, established methods for second-order quantifier elimination might be applied to process logic programs, which might be especially interesting for nonground programs. On the other hand, known efficient techniques for processing logic programs with specific semantics get embedded in a wider context when seen as particular efficient second-order quantifier elimination methods for constrained inputs.

Acknowledgements. I am obliged to anonymous referees of an earlier version for suggestions to improve the presentation and bringing important related works [Mar92, Sch95, Jan06] to attention.

References

[Apt88] K. R. Apt, H. A. Blair, and A. Walker. Towards a theory of declarative knowledge. In Jack Minker (ed.), *Foundations of deductive databases and logic programming*, pp. 89–148. Morgan Kaufmann, San Francisco, 1988.

[Cla78] K. L. Clark. Negation as failure. In H. Gallaire and J. Minker (eds.), *Logic and Databases*, pp. 292–322. Plenum Press, New York, 1978.

[Ebb84] H.-D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic*. Springer, New York, 1984.

[Eit08] T. Eiter and K. Wang. Semantic forgetting in answer set programming. *Artif. Int.*, 172:1644–1672, 2008.

[Fer07] P. Ferraris, J. Lee, and V. Lifschitz. A new perspective on stable models. In *IJCAI-07*, pp. 372–379. 2007.

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

[Gab08] D. M. Gabbay, R. A. Schmidt, and A. Szałas. *Second-Order Quantifier Elimination: Foundations, Computational Aspects and Applications*. College Publications, London, 2008.

[Geb09] M. Gebser, B. Kaufmann, and T. Schaub. Solution enumeration for projected Boolean search problems. In *CPAIOR 2009*, pp. 71–86. 2009.

[Gel88] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP 1988*, pp. 1070–1080. 1988.

[Ino98] K. Inoue and Chiaki Sakama. Negation as failure in the head. *J. of Logic Prog.*, 35(1):39–78, 1998.

[Jan06] T. Janhunen, I. Niemelä, D. Seipel, Patrik Simons, and Jia-Huai You. Unfolding partiality and disjunctions in stable model semantics. *ACM Trans. on Comput. Logic*, 7(1):1–37, 2006.

[Lee06] J. Lee and F. Lin. Loop formulas for circumscription. *Artificial Intelligence*, 170:160–185, 2006.

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

[Lif08] V. Lifschitz. Twelve definitions of a stable model. In *ICLP 2008*, pp. 37–51. 2008.

[Lin91] F. Lin. *A Study of Nonmonotonic Reasoning*. Ph.D. thesis, Stanford University, 1991.

[Mar92] W. Markek and V. S. Subrahmanian. The relationship between stable, supported, default and autoepistemic semantics for general logic programs. *Theor. Computer Science*, 103:365–386, 1992.

[McC80] John McCarthy. Circumscription – a form of non-monotonic reasoning. *Artif. Int.*, 13:27–39, 1980.

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

[Wer08] C. Wernhard. Literal projection for first-order logic. In *JELIA 08*, pp. 389–402. 2008.

[Wer09] C. Wernhard. Tableaux for projection computation and knowledge compilation. In *TABLEAUX 2009*, pp. 325–340. 2009.

[Wer10a] C. Wernhard. Circumscription and projection as primitives of logic programming – extended version. Tech. rep., TU Dresden, 2010. Available from http://cs.christophwernhard.com/papers/logprog2010extended.pdf.

[Wer10b] C. Wernhard. Literal projection and circumscription. In *FTP'09*, *CEUR Proc.*, vol. 556. 2010.