Knowl Inf Syst (2010) 25:137–168
DOI 10.1007/s10115-010-0300-5

REGULAR PAPER

Hybrid rules with well-founded semantics

Włodzimierz Drabent · Jan Małuszyński

Received: 16 November 2007 / Revised: 13 March 2009 / Accepted: 27 March 2010 /
Published online: 23 May 2010

Abstract A general framework is proposed for integration of rules and external first-order theories. It is based on the well-founded semantics of normal logic programs and inspired by ideas of Constraint Logic Programming (CLP) and constructive negation for logic programs. Hybrid rules are normal clauses extended with constraints in the bodies; constraints are certain formulae in the language of the external theory. A hybrid program consists of a set of hybrid rules and an external theory. Instances of the framework are obtained by specifying the class of external theories and the class of constraints. An example instance is integration of (non-disjunctive) Datalog with ontologies formalized in description logics. The paper defines a declarative semantics of hybrid programs and a goal-driven formal operational semantics. The latter can be seen as a generalization of SLS-resolution. It provides a basis for hybrid implementations combining Prolog with constraint solvers (such as ontology reasoners). Soundness of the operational semantics is proven. Sufficient conditions for decidability of the declarative semantics and for completeness of the operational semantics are given.

Keywords Integration of rules and ontologies · Semantic web reasoning · Knowledge representation · Well-founded semantics · Constructive negation · Constraint logic programming

W. Drabent (✉)
Institute of Computer Science, Polish Academy of Sciences,
ul. Ordona 21, 01-237 Warszawa, Poland
e-mail: wdr@ida.liu.se

W. Drabent · J. Małuszyński
Department of Computer and Information Science,
Linköping University, 581 83 Linköping, Sweden

J. Małuszyński
e-mail: jmz@ida.liu.se

1 Introduction

This paper presents an approach to integration of normal logic programs under the well-founded semantics with external first-order theories. The problem is motivated by the discussion about the rule level of the Semantic Web.

It is often claimed that rule-based applications need non-monotonic reasoning for handling negative information. As the issue of non-monotonic reasoning and negation was thoroughly investigated in the context of logic programming (see e.g. [3] for a survey of classical work on this topic), it would be desirable to build-up on this expertise. Well-established formal semantics: the answer set semantics [5], the well-founded semantics [40], and the 3-valued completion semantics of Kunen [25] provide theoretical foundations for existing logic programming systems. On the other hand, applications refer usually to domain-specific knowledge, that is often supported by specific reasoning or computational mechanisms.

Domain-specific variants of logic programs are handled within the constraint logic programming framework CLP [30]. In CLP, the concept of constraint domain makes it possible to extend the semantics of pure logic programs and to use domain-specific constraint solvers for sound reasoning. Classical CLP does not support non-monotonic reasoning, but integration of both paradigms is discussed by some researchers (see e.g. [9,20,39]). Special kinds of domain-specific knowledge are domain-specific terminologies, specified in a formal ontology description language, such as OWL [33]. This raises the issue of integration of rules and ontologies, which has achieved a considerable attention (see e.g. [7,17,31,32,35] and references therein; an overview can be found in Refs. [1,13,28]). It is commonly assumed that an ontology is specified as a set of axioms in (a subset of) First-Order Logic (FOL), usually in a Description Logic. A rule language, typically a variant of Datalog, is then extended by allowing a restricted use of ontology predicates. The extensions considered in the literature are mostly based on disjunctive Datalog with negation under the answer set semantics.

In contrast to that, our focus is on normal logic programs under the well-founded semantics. Our objective is to extend them in such a way that domain-specific knowledge represented by a first-order theory can be accessed from the rules. The theory will be called the external theory. Going beyond Datalog makes it possible to use data structures, like lists, for programming in the extended rule language. We want to define the semantics of the extended language so that the existing reasoners for normal logic programs and for the external theory can be re-used for querying the extended programs. Thus, our objective is to provide a framework for hybrid integration of normal logic programs and external theories. Integration of Datalog rules with ontologies specified in a Description Logic could then be handled within the framework as a special case.

The choice of the well-founded semantics as the basis for our approach is motivated by existence of top-down query answering algorithms, which facilitate building a query answering method for our framework. Notice also that the well-founded semantics and the answer set semantics are equivalent for a wide class of programs, including stratified normal programs.

We introduce a notion of hybrid program; such a program is a pair $(P, \mathcal{T})$ where $\mathcal{T}$ is a set of axioms in a first-order language $\mathcal{L}$ and P is a set of *hybrid rules*. $\mathcal{T}$ and P share function symbols but have disjoint alphabets of predicate symbols. This reflects the intuition that domain-specific knowledge is shared by many applications and is not redefined by the applications. A hybrid rule is a normal clause whose body may include a formula in $\mathcal{L}$, called *the constraint* of the rule. We define declarative semantics of hybrid programs as a natural extension of the well-founded semantics of logic programs. It is 3-valued; the semantics of a program $(P, \mathcal{T})$ is a set of ground literals over the alphabet of P. The semantics is

undecidable (as is the well-founded semantics for normal programs). However, it is decidable for Datalog hybrid programs, provided the constraints are decidable.

The operational semantics presented in this paper is goal driven, and allows non-ground goals. It employs data structures built of possibly non-ground goals and constraints. It combines a variant of SLS-resolution (see e.g. [3]) with handling constraints. The latter includes checking satisfiability of the constraints w.r.t. $\mathcal{T}$, which is assumed to be done by a reasoner of $\mathcal{T}$. Thus the operational semantics provides a basis for development of implementations integrating LP (logic programming) reasoners supporting/approximating the well-founded semantics (such as XSB Prolog [37]) with constraint solvers (such as ontology reasoners). The operational semantics is sound w.r.t. the declarative one, under rather weak sufficient conditions. It is complete for Datalog hybrid programs under a certain syntactic condition of safeness.

In the special case of hybrid rules without non-monotonic negation, the rules can be seen as the usual implications of the FOL, thus as additional axioms extending $\mathcal{T}$. In this case, every ground atom which is true in the semantics of the hybrid program is a (2-valued) logical consequence of $P \cup \mathcal{T}$. The paper is organized as follows. Section 2 gives an introduction to the well-founded semantics of normal logic programs and presents the notion of constraint used in this paper. Basic ideas of Description Logics and their use for defining ontologies and ontological constraints are briefly discussed. The proposed approach to combining LP with external theories is informally introduced by means of examples. Section 3 gives a formal presentation of the syntax and declarative semantics of the generic language of hybrid rules, parameterized by the constraint domain. Section 4 introduces the operational semantics; then soundness and completeness results relating the declarative semantics and the operational semantics are stated and proven. Section 5 discusses decidability of the declarative semantics. The last two sections contain discussion of related work and conclusions. A preliminary, abbreviated version of this work appeared as [16].

2 Preliminaries

2.1 Normal logic programs and the well-founded semantics

In this work, we use the standard terminology and notation of logic programming (see e.g. [2]).

The language of hybrid rules will be defined as an extension of normal logic programs. We assume that the programs are built over a first-order alphabet including a set $\mathcal{P}_R$ of predicates, a set $\mathcal{V}$ of variables and a set $\mathcal{F}$ of function symbols with their arities, including a non-empty set of symbols of arity 0, called *constants*.

Atomic formulae (or *atoms*) and *terms* are built in a usual way. A *literal* is an atomic formula (*positive literal*) or a negated atomic formula (*negative literal*). A literal (a term) not including variables is called *ground*.

A **normal logic program** P is a finite set of rules of the form

$$H \leftarrow B_1, \ldots, B_n \quad \text{where } n \geq 0$$

where H is an atomic formula, and $B_1, \ldots, B_n$ are literals. The rules are also called *normal clauses*. The rules with empty bodies ($n = 0$) are called *facts* or *unary clauses*; they are usually written without $\leftarrow$. A normal clause is called *definite clause* iff all literals of its body are positive. A *definite program* is a finite set of definite clauses. In this paper, a **Datalog** program is a normal logic program with $\mathcal{F}$ being a finite set of constants.

The **Herbrand base** $\mathcal{H}$ is the set of all ground atoms built with the predicates from $\mathcal{P}_R$, and function symbols from $\mathcal{F}$. For a subset $S \subseteq \mathcal{H}$, by $\neg S$ we denote the set of negations of the elements of S, $\neg S = \{\neg a \mid a \in S\}$. A ground instance of a rule R is a rule R' obtained by replacing each variable of R by a ground term over the alphabet. The set of all ground instances of the rules of a program P will be denoted $ground(P)$. Notice that in the case of Datalog $ground(P)$ is a finite set of ground rules.

A *3-valued Herbrand interpretation* (briefly – **interpretation**) $\mathcal{I}$ of P is a subset of $\mathcal{H} \cup \neg\mathcal{H}$ such that for no ground atom A both A and $\neg A$ are in $\mathcal{I}$. Intuitively, the set $\mathcal{I}$ assigns the truth value **t** (true) to all its members. Thus, A is false (has the truth value **f**) in $\mathcal{I}$ iff $\neg A \in \mathcal{I}$, and $\neg A$ is false in $\mathcal{I}$ iff $A \in \mathcal{I}$. If $A \notin \mathcal{I}$ and $\neg A \notin \mathcal{I}$ then the truth value of A (and that of $\neg A$) is **u** (undefined). This is in a natural way generalized to non-ground atoms and non-atomic formulae (see e.g. [3]). An interpretation $\mathcal{I}$ is a model of a formula F (which is denoted by $\mathcal{I} \models_3 F$) iff F is true in $\mathcal{I}$.

As usual, a *2-valued Herbrand interpretation* is a subset of $\mathcal{H}$. It assigns the value **t** to all its elements and the value **f** to all remaining elements of the Herbrand base. It is well known that any definite program P has a unique least[1] 2-valued Herbrand model. We will denote it $\mathcal{M}_P$. A normal program may not have the least Herbrand model.

The well-founded semantics of logic programs [40] assigns to every program P a unique (three valued) Herbrand model, called the *well-founded* model of P. Intuitively, the facts of a program should be true, and the ground atoms which are not instances of the head of any rule should be false. This information can be used to reason which other atoms must be true and which must be false in any Herbrand model. Such a reasoning gives in the limit the well-founded model, where the truth values of some atoms may still be undefined. The well-founded semantics has several equivalent formulations. We briefly sketch here a definition following that of Ref. [21].

While defining the well-founded model, for every predicate symbol p, we will treat $\neg p$ as a new distinct predicate symbol. A normal program can thus be treated as a definite program over Herbrand base $\mathcal{H} \cup \neg\mathcal{H}$. A 3-valued interpretation over $\mathcal{H}$ can be treated as a 2-valued interpretation over $\mathcal{H} \cup \neg\mathcal{H}$.

Let I be such an interpretation ($I \subseteq \mathcal{H} \cup \neg\mathcal{H}$). We define two ground, possibly infinite, definite programs $P/_tI$ and $P/_{tu}I$. For a given program P, $P/_tI$ is the ground instantiation of P together with ground unary clauses that show which negative literals are true in I.

$$P/_tI = ground(P) \cup \{\neg A \mid \neg A \in I\}$$

$P/_{tu}I$ is similar but all the negative literals that are true or undefined in I are made true here:

$$P/_{tu}I = ground(P) \cup \{\neg A \mid A \notin I,\ A \in \mathcal{H}\}$$

Now we define an operator $\Psi_P(I)$ which produces a new Herbrand interpretation of P:

$$\Psi_P(I) = (\mathcal{M}_{P/_tI} \cap \mathcal{H}) \cup \neg(\mathcal{H} \backslash \mathcal{M}_{P/_{tu}I})$$

It can be proved that the operator is monotonic; $\Psi_P(I) \subseteq \Psi_P(J)$ whenever $I \subseteq J$. Its least fixed point is called the **well-founded model** $WF(P)$ of program P. For some countable ordinal α we have $WF(P) = \Psi_P^\alpha(\emptyset)$.

The following example shows a simple Datalog program and its well-founded model.

Example 2.1 A two person game consists in moving a token between vertices of a directed graph. Each move consists in traversing one edge from the actual position. Each of the players

[1] in the sense of set inclusion.

makes one move in order. The graph is described by a database of facts $m(X, Y)$ corresponding to the edges of the graph. A position X is said to be a *winning position* X if there exists a move from X to a position Y which is a losing (non-winning) position:

$$w(X) \leftarrow m(X, Y), \neg w(Y).$$

Consider the graph

$$\begin{array}{ccccccc} & & & & d & \rightarrow & e \\ & & & & \uparrow & & \downarrow \\ b & \leftrightarrow & a & \rightarrow & c & \rightarrow & f \end{array}$$

and assume that it is encoded by the facts $m(b, a), m(a, b), \ldots, m(e, f)$ of the program. The winning positions are e, c. The losing positions are d, f. Position a is not a losing one since the player has an option of moving to b from which the partner can only return to a. This intuition is properly reflected by the well-founded model of the program, it contains the following literals with the predicate symbol w: $w(c), w(e), \neg w(d), \neg w(f)$.

This example follows [22,40]. A non-Datalog version, with an infinite graph, is presented in Ref. [11].

2.2 External theories

In this section, we discuss logical theories to be integrated with logic programs.

2.2.1 Constraints

Our objective is to define a general framework for extending normal logic programs, which, among others, can also be used for integration of Datalog rules with ontologies. Syntactically, the clauses of a logic program are extended by adding certain formulae of a certain logical theory. The added formulae will be called **constraints**. We use this term due to similarities with constraint logic programming [30].

We will consider a 2-valued FOL theory, called *external theory* or *constraint theory*. A set of its formulae is chosen as the set of constraints.[2] The function symbols and the variables of the language of the external theory are the same as those of the language of rules. On the other hand, the predicate symbols of both languages are distinct. We will call them *constraint predicates* and *rule predicates*. We assume that the external theory is given by a set of axioms $\mathcal{T}$ and the standard consequence relation $\vdash$ of the FOL, or equivalently the logical consequence $\models$. (Other consequence relations can be used instead; for instance deriving those formulae which are true in a canonical model of $\mathcal{T}$, or in a given class of models). We will sometimes use $\mathcal{T}$ as the name of the theory.

Sometimes one deals with an external theory whose set $\mathcal{F}_c$ of function symbols is a proper subset of the set $\mathcal{F}$ of function symbols of the rules. For instance, the external theory uses only constants, and the rules employ term constructors (i.e. non-constant function symbols). In such case, we simply extend the alphabet of the external theory so that its set of function symbols is $\mathcal{F}$. The modified external theory is a conservative extension of the original one [38]. A formula without symbols from $\mathcal{F} \backslash \mathcal{F}_c$ is a logical consequence of $\mathcal{T}$ in one of them iff it is a logical consequence in the other. Thus, such modification of the external theory is inessential; this justifies our assumption of a common alphabet of function symbols.

[2] Our operational semantics imposes certain restrictions on the set of constraints. They are discussed in Sect. 4.1. The declarative semantics works for arbitrary constraints.

2.2.2 Ontologies and ontological constraints

This section surveys some basic concepts of Description Logics (DLs) [4] and the use of DLs for specifying ontologies. An *ontology* may be defined as a "specification of a conceptualization" [23]. An ontology should thus provide a formal definition of the terminology to be shared.

Description Logics are specific fragments of the FOL. The syntax of a DL is built over disjoint alphabets of *class names*, *property names* and *individual names*. From the point of view of FOL they are, respectively, one and two argument predicate symbols, and constants. Depending on the kind of DL, different constructors are provided to build class expressions (or briefly *classes*) and property expressions (or briefly *properties*). Some DLs allow also to represent concrete datatypes, such as strings or integers. In that case one distinguishes between individual-valued properties and data-valued properties.

By an ontology we mean a finite set of axioms in some decidable DL. The axioms describe classes and properties of the ontology and assert facts about individuals and data. An ontology is thus a DL knowledge base consisting of two parts: a *TBox* (terminology) including class axioms and property axioms and an *ABox* (assertions) stating facts about individuals and data. The axioms of DLs can be seen as an alternative representation of FOL formulae. Thus, the semantics of DLs is defined by referring to the usual notions of interpretation and model, and an ontology can be considered a FOL theory.

For most of decidable DLs, there exist well-developed automatic reasoning techniques. Given an ontology $\mathcal{T}$ in a DL one can use a respective reasoner for checking if a formula C is a logical consequence of $\mathcal{T}$. If $\mathcal{T} \not\models C$ and $\mathcal{T} \not\models \neg C$ then C is true in some models of the ontology and false in some other models.

Ontologies are often specified in the standard web ontology language OWL DL, based on the Description Logic $\mathcal{SHOIN}(D)$. OWL Ontologies can be seen as sets of axioms in this DL.

OWL DL class axioms make it possible to state class equivalence $A \equiv C$ and class inclusion $A \sqsubseteq C$, where A is a class name and C is a class expression. Class expressions are built from class names using constructors, such as $\top$ (the universal concept), $\bot$ (the bottom concept), intersection, union and complement. Classes can also be described by direct enumeration of members and by *restrictions* on properties (for more details see [33]).

Property axioms make it possible to state inclusion of properties, specify the domain and the range of a property, state that a property is symmetric, transitive, functional, or inverse functional.

OWL DL assertions indicate members of classes and properties. Individuals are referred to by individual names. It is possible to declare that given individual names represent the same individual or that each of them represents a different individual.

The following example using some expressive constructions of OWL DL will be used in the sequel to discuss how integration of Datalog with OWL DL ontologies is achieved in our framework.

Example 2.2 In some research area, an author of at least 3 books is considered an expert. An OWL DL ontology referring to this research area has classes $Author$ and $Book$, and a property *isAuthorOf* with domain $Author$ and range $Book$. The class $Expert$ can now be defined using OWL DL cardinality restriction:[3]

$$Expert \equiv isAuthorOf \ \textbf{min}\ 3\ Book$$

[3] In the Manchester OWL Syntax http://www.co-ode.org/resources/reference/manchester_syntax/.

The property *isAuthorOf* has the inverse property *hasAuthor*. The following class expression defines the class of authors which co-authored a book with a given author X (e.g. $smith$)

$$(isAuthorOf \ \textbf{some} \ (hasAuthor \ \textbf{value} \ X))$$

All individuals of class $Book$ which appear in the ontology are declared as distinct. The ontology states that the individuals *johns* and $brown$ of class $Author$ are the same. (This may happen e.g. due to a change of the name of a person). There are also authors $smith$ and $burns$; $smith$, $burns$ and *johns* are declared to be distinct. In addition, the ontology asserts that *johns* is an author of the books $b1$, $b2$ and $brown$ is an author of the books $b2$, $b3$. Thus an OWL DL reasoner will conclude that *johns* ($brown$) is an expert.

2.3 Datalog with constraints: introductory examples

We now illustrate the idea of adding constraints to rule bodies on two simple examples. The intention is to give an informal introduction to the semantics of hybrid rules. The first example will be used later on to accompany the formal presentation of the declarative and operational semantics of our framework. The second one illustrates some aspects of expressing external theories in OWL DL.

Example 2.3 The example describes a variant of the game from Example 2.1 where the rules are subject to additional restrictions. Assume that the positions of the graph represent geographical locations described by an ontology. The ontology provides, among others, the following information

- subclass relations (TBox axioms): e.g. $Fi \sqsubseteq E$ (locations in Finland are locations in Europe);
- classification of some given locations represented by constants (ABox axioms). For instance, assuming that the positions of Example 2.1 represent locations we may have $Fi(b)$ (b is a location in Finland), $E(c)$ (c is a location in Europe).

We now add some restrictions as ontological constraints[4] added to the facts $m(e, f)$ and $m(c, f)$:

$$w(X) \leftarrow m(X, Y), \neg w(Y)$$

$$m(b, a) \qquad m(c, f) \leftarrow \neg Fi(f)$$

$$m(a, b) \qquad m(e, f) \leftarrow E(f)$$

$$m(a, c)$$

$$m(c, d)$$

$$m(d, e)$$

$$b \leftrightarrow a \rightarrow c \xrightarrow[\neg Fi(f)]{} f \qquad d \longrightarrow e \quad (\downarrow E(f))$$

Intuitively, this would mean that the move from e to f is allowed only if f is in Europe and the move from c to f—only if f is not in Finland. These restrictions may influence the outcome of the game: f will still be a losing position but if the axioms of the ontology do not allow to conclude that f is in Europe, we cannot conclude that e is a winning position. However, we can conclude that if f is not in Europe then it cannot be in Finland. Thus, at least one of the conditions $E(f)$, $\neg Fi(f)$ holds. If $E(f)$ then, as in Example 2.1, e is a winning position, d is a losing one, hence c is a winning position. On the other hand, if $\neg Fi(f)$ then the move from c to f is allowed, in which case c is a winning position. Therefore c is always a winning position; $w(c)$ is considered to be a consequence of the program.

[4] Symbol $\neg$ is used to denote two kinds of negation. Within a constraint it is the classical negation of the external theory. When applied to a rule predicate, $\neg$ denotes non-monotonic negation. Thus two distinct negation symbols are not needed.

Example 2.4 A committee of reviewers is to be created for evaluation of the applicants for an opened position. A reviewer has to be an expert, as defined by the ontology of Example 2.2 and must not have a conflict of interest (*coi*) with an applicant. Persons who are co-authors of a book have *coi*. (This implies that an author of a book has *coi* with himself/herself; this applies in particular to each expert). Additionally, some conflicts of interest are declared by facts.

The following rules define a potential reviewer X for a candidate Y (relation $mayreview$). Two constraints are used: $Expert(X)$ and (*isAuthorOf* **some** (*hasAuthor* **value** X))(Y). They refer to the ontology of Example 2.2.

$$mayreview(X, Y) \leftarrow Expert(X), \neg coi(X, Y)$$
$$coi(X, Y) \leftarrow (isAuthorOf \textbf{ some } (hasAuthor \textbf{ value } X))(Y)$$
$$coi(johns, burns)$$

The intention is to query the rules and the ontology for checking if a given person may be a reviewer for a given candidate. Consider the individual *johns* of Example 2.2 and check if she might be appointed a reviewer for some of the people named in the ontology. An OWL DL reasoner can check that *johns* is an expert and that she has the conflict of interest with herself, i.e. with *johns* alias $brown$. The conflict of interest with $burns$ is stated explicitly. So *johns* cannot be appointed a reviewer for herself and for $burns$.

To check if *johns* has the conflict of interest with $smith$ one has to refer to the ontology for checking if they co-authored a book. If this is confirmed by the reasoner (e.g. when the ontology asserts that both *johns* and $smith$ are authors of $b1$) then $coi(johns, smith)$ is true and *johns* cannot be a reviewer for $smith$. If non-existence of any co-authored book follows from the ontology, then $coi(johns, smith)$ is false and *johns* can be a reviewer. Otherwise[5] *johns* may be a reviewer for $smith$ under the condition that they did not co-authored a book. This constraint should be returned in the answer to the query.

An example employing non-nullary function symbols is given in Ref. [14]. The semantics of hybrid programs presented below formalizes the intuitions presented in the examples of this section.

3 Integration of rules and external theories

This section defines the syntax and the (declarative) semantics of hybrid programs, integrating normal rules with first-order theories. The general principles discussed here apply in a special case to integration of Datalog with ontologies specified in Description Logics.

3.1 Syntax

We consider a first-order alphabet including, as usual, disjoint alphabets of predicate symbols $\mathcal{P}$, function symbols $\mathcal{F}$ (including a set of constants) and variables $\mathcal{V}$. We assume that $\mathcal{P}$ consists of two disjoint sets $\mathcal{P}_R$ (*rule predicates*) and $\mathcal{P}_C$ (*constraint predicates*). The atoms and the literals constructed with these predicates will respectively be called *rule atoms* (*rule literals*) and *constraint atoms* (*constraint literals*). We will combine rules over alphabets $\mathcal{P}_R$, $\mathcal{F}$, $\mathcal{V}$ with an external theory $\mathcal{T}$ over $\mathcal{P}_C$, $\mathcal{F}$, $\mathcal{V}$, employing constraints (a distinguished set of formulae of $\mathcal{T}$).

[5] They are co-authors in some models of the ontology, and are not in some others.

Definition 3.1 A **hybrid rule** (over $\mathcal{P}_R,\ \mathcal{P}_C,\ \mathcal{F},\ \mathcal{V}$) is an expression of the form:

$$H \leftarrow C, L_1, \ldots, L_n$$

where, $n \geq 0$ each L_i is a rule literal and C is a constraint (over $\mathcal{P}_C,\ \mathcal{F},\ \mathcal{V}$); C is called the **constraint of the rule**.

A **hybrid program** is a pair $(P, \mathcal{T})$ where P is a set of hybrid rules and $\mathcal{T}$ is a set of axioms over $\mathcal{P}_C,\ \mathcal{F},\ \mathcal{V}$.

Hybrid rules are illustrated in Example 2.3. We adopt a convention that a constraint **true**, which is a logical constant interpreted as **t**, is omitted. Usually we do not distinguish between sequences, like $L_1, \ldots, L_n$, and conjunctions, like $L_1 \wedge \cdots \wedge L_n$. Notation $\overline{L}$ will be used to denote a sequence of rule literals (similarly $\overline{t}$ a sequence of terms, etc.); $\overline{t} = \overline{u}$ will denote a conjunction of equalities $t_1 = u_1, \ldots, t_k = u_k$.

3.2 Declarative semantics

The declarative semantics of hybrid programs is defined as a generalization of the well-founded semantics of normal programs; it refers to the models of the external theory $\mathcal{T}$ of a hybrid program. Given a hybrid program $(P, \mathcal{T})$, we cannot define a unique well-founded model of P since we have to take into consideration the logical values of the constraints in the rules. However, a unique well-founded model can be defined for any given model of $\mathcal{T}$. Roughly speaking, the constraints in the rules are replaced by their logical values in the model (**t** or **f**); then the well-founded model of the obtained logic program is taken. The well-founded models are over the Herbrand universe, but the models of $\mathcal{T}$ are arbitrary.

By applying a substitution $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ to a formula F we mean applying it to the free variables of F. Moreover, if a bound variable x of F occurs in some t_i $(1 \leq i \leq n)$ then x in F is replaced by a new variable.

By a **ground instance** of a hybrid rule $H \leftarrow C, L_1, \ldots, L_n$, where C is the constraint of the rule, we mean any rule $H\theta \leftarrow C\theta, L_1\theta, \ldots, L_n\theta$, where θ is a substitution replacing the free variables of $H \leftarrow C, L_1, \ldots, L_n$ by ground terms (over the alphabet $\mathcal{F}$). So the constraint $C\theta$ has no free variables, and $H\theta, L_1\theta, \ldots, L_n\theta$ are ground literals. By $ground(P)$ we denote the set of all ground instances of the hybrid rules in P.

Definition 3.2 Let $(P, \mathcal{T})$ be a hybrid program and let M_0 be a model of $\mathcal{T}$. Let P/M_0 be the normal program obtained from $ground(P)$ by

- removing each rule constraint C which is true in M_0 (i.e. $M_0 \models C$),
- removing each rule whose constraint C is not true in M_0, (i.e. $M_0 \not\models C$).

The well-founded model $WF(P/M_0)$ of P/M_0 is called the **well-founded model** of P *based* on M_0.

A formula F (over $\mathcal{P}_R, \mathcal{F}, \mathcal{V}$) **holds** (is *true*) in the well-founded semantics of a hybrid program $(P, \mathcal{T})$ (denoted $(P, \mathcal{T}) \models_{\text{wf}} F$) iff $M \models_3 F$ for each well-founded model M of $(P, \mathcal{T})$. □

Notice that the negation in the rule literals is non-monotonic, and the negation in the constraints is that from the external theory, thus monotonic.

We say that F is *false* in the well-founded semantics of $(P, \mathcal{T})$ if $(P, \mathcal{T}) \models_{\text{wf}} \neg F$, and that F is *undefined* if the logical value of F in each well-founded model of $(P, \mathcal{T})$ is **u**. There is a fourth case: F has distinct logical values in various well-founded models of P. Formally, the semantics of $(P, \mathcal{T})$ does not assign any truth value to such F. We may say

that its truth value depends on the considered model of the external theory. Classes of models in which F has a specific truth value can be characterized by constraints. Such constraints provide sufficient conditions for F to have the specific truth value. They are constructed by the proposed operational semantics.

Example 3.3 For the hybrid program $(P, \mathcal{T})$ of Example 2.3, we have to consider models of the ontology $\mathcal{T}$. For every model M_0 of $\mathcal{T}$ such that $M_0 \models E(f)$ the program P/M_0 includes the fact $m(e, f)$. The well-founded model of P/M_0 includes thus the literals $\neg w(f)$, $w(e)$, $\neg w(d)$, $w(c)$ (independently of whether $M_0 \models Fi(f)$).

On the other hand, for every model M_1 of the ontology such that $M_1 \models \neg Fi(f)$ the program P/M_1 includes the fact $m(c, f)$. The well-founded model of P/M_1 includes thus the literals $\neg w(f)$, $w(c)$ (independently of whether $M_1 \models E(f)$).

Notice that each of the models of the ontology falls in one of the above discussed cases. Thus, $w(c)$ and $\neg w(f)$ hold in the well-founded semantics of the hybrid program, and the logical value of $w(a)$ and that of $w(b)$ is $\mathbf{u}$ in each well-founded model of the program. On the other hand, $w(e)$ and $\neg w(d)$ are true in those well-founded models $WF(P/M_0)$ of P for which the constraint $E(f)$ is true in M_0. Similarly, $\neg w(e)$ and $w(d)$ are true in those models for which $E(f)$ is false. Thus the well-founded semantics assigns unique truth values to $w(a)$, $w(b)$, $w(c)$ and $w(f)$, but not to $w(d)$ and $w(e)$. The truth values of $w(d)$ and $w(e)$ can be characterized by additional constraints.

Consider a case of hybrid rules without negative rule literals. So the non-monotonic negation does not occur. Such rules can be seen as implications of FOL and treated as axioms added to $\mathcal{T}$. For such case the well-founded semantics is *compatible with FOL* in the following sense: For any ground rule atom A if $(P, \mathcal{T}) \models_{\mathrm{wf}} A$ then $P \cup \mathcal{T} \models A$.[6] We omit a detailed proof.

As the well-founded semantics of normal programs is undecidable, so is the well-founded semantics of hybrid programs. It is, however, decidable for Datalog hybrid programs with decidable external theories (Sect. 5). In Sect. 4, we show that sound reasoning is possible (for arbitrary hybrid programs) by appropriate generalization of SLS-resolution. For the Datalog case, the proposed reasoning scheme is complete under a certain safeness condition.

3.3 Treatment of equality

In this section, we discuss how equality is treated by the declarative semantics introduced earlier. The semantics is based on Herbrand models. Thus, it treats distinct ground terms as having different values.

Example 3.4 Consider a hybrid program $(P, \mathcal{T})$, where $P = \{\, p(a) \,\}$. Both $p(a)$ and $\neg p(b)$ hold in the well-founded semantics of $(P, \mathcal{T})$, even if $\mathcal{T}$ implies that a and b are equal. This feature of the semantics of hybrid programs may be found undesirable.

We will call this phenomenon the *problem of two equalities*. Below we first show that the problem is well known in constraint logic programming (CLP) and explain how it is dealt with. Then we discuss two more formal ways of avoiding it: external theories where equality

[6] The reverse implication does not hold. As a counterexample take $\mathcal{T} = \{\exists x.q(x)\}$ and $P = \{\, p \leftarrow q(x), r(x);\ r(x) \leftarrow \}$. $P \cup \mathcal{T} \models p$ but $(P, \mathcal{T}) \not\models_{\mathrm{wf}} p$, as there exist models of $\mathcal{T}$ in which each ground atom $q(t)$ is false.

We can obtain (something close to) the reverse implication by considering only those well-founded models which are based on Herbrand models of $\mathcal{T}$. If $P \cup \mathcal{T} \models A$ then $M \models A$ for each well-founded model M of P based on a Herbrand interpretation M_0 of $\mathcal{T}$.

satisfies Clark equality theory (CET), and hybrid rules which are congruent w.r.t. a given external theory.

The problem of two equalities is familiar from CLP [30], and is not found troublesome in practice. Most CLP implementations employ both syntactic equality and equality of the constraint domain.[7] Let us denote the latter by $='$ (and use $=$ for the syntactic equality of the Herbrand domain). Formally, let us treat $=$ as equality, and $='$ as an equivalence relation. As an example consider CLP over arithmetic constraints [30]. Terms $2+2$ and 4 are distinct but denote the same number, we have $2+2 \neq 4$ and $2+2 =' 4$. Constraint predicates (such as arithmetic inequality $<$) treat $2+2$ and 4 as equal (Formally, $='$ is a congruence for the constraint predicates: $p(t_1, \ldots, t_n)$ iff $p(u_1, \ldots, u_n)$ whenever $t_1 =' u_1, \ldots, t_n =' u_n$, for any constraint predicate p). Other predicates may distinguish such terms. This is related to using unification in the operational semantics; unification is related to the syntactic equality.

Apparently the programmers find this feature natural and not confusing. They are aware of dealing both with the Herbrand interpretation and with a non-Herbrand one. They know that the latter is employed only by constraint predicates. They take care of distinguishing the two corresponding equalities. For instance, to express a fact that $size$ should be true for the number 4, a rule $size(N) \leftarrow N =' 4$ will be used (instead of a fact $size(4)$.).

Assume that we have an external theory $\mathcal{T}$ with equality $='$. We say that a set of hybrid rules P is **congruent** for a predicate symbol p w.r.t. $\mathcal{T}$ when $\mathcal{T} \models t_1 =' u_1, \ldots, t_n =' u_n$ implies

$$(P, \mathcal{T}) \models_{\mathrm{wf}} p(t_1, \ldots, t_n) \quad \text{iff} \quad (P, \mathcal{T}) \models_{\mathrm{wf}} p(u_1, \ldots, u_n),$$

for any ground terms $t_1, \ldots, t_n, u_1, \ldots, u_n$. When P is congruent w.r.t. $\mathcal{T}$ for any rule predicate p then we say that P is congruent w.r.t. $\mathcal{T}$ (or shortly that $(P, \mathcal{T})$ is congruent).

Example 3.5 Program $P = \{ p(a) \}$ (from Example 3.4) is not congruent w.r.t. any $\mathcal{T}$ in which $\mathcal{T} \models a =' b$.

The hybrid program from Example 2.4 with the fact $coi(johns, burns)$ removed is congruent,[8] independently from $\mathcal{T}$.

Example 3.6 (Constructing congruent programs) Consider the program from Examples 2.3, 3.3. The program implies $w(c)$ and $\neg w(g)$ (formally $(P, \mathcal{T}) \models_{\mathrm{wf}} w(c)$ and $(P, \mathcal{T}) \models_{\mathrm{wf}} \neg w(g)$). Assume that $\mathcal{T}$ implies that $c =' g$. For instance, the equality may be explicitly stated by an `owl:sameAs` assertion. Informally, equality $c =' g$ is incompatible with P; the rules of P treat differently the objects c, g, while $\mathcal{T}$ states that they are equal. Formally, $(P, \mathcal{T})$ is not congruent.

One can modify P to make it treat c, g in the same way. It is sufficient to add rules $m(a, g)$, $m(g, d)$, and $m(g, f) \leftarrow \neg Fi(f)$. (We replace c by g in the rules of P). Now $w(c)$ and $w(g)$ hold in the well-founded semantics of the obtained program $(P', \mathcal{T})$. The program is congruent, provided that $\mathcal{T}$ does not imply $t_1 =' t_2$ for any other pair $\{t_1, t_2\} \neq \{c, g\}$ of constants occurring in the program.

We can modify P to make it congruent independently from $\mathcal{T}$. The idea is to replace (implicit) $=$ by explicit $='$. For instance we may replace in P the rule $w(X) \leftarrow m(X, Y), \neg w(Y)$ by

$$w(X) \leftarrow X =' X',\ Y =' Y',\ m(X', Y'),\ \neg w(Y).$$

[7] See for instance the comment on an example constraint domain on p. 414 in [30, Section 12.2].

[8] The unchanged program is not congruent, unless the ontology implies that *johns* (*brown*) co-authored a book with *burns*. This is because $coi(johns, burns)$ holds and $coi(brown, burns)$ does not hold in the well-founded semantics of the program, but $johns =' brown$.

The obtained program $(P'', \mathcal{T})$ is congruent for w, w.r.t. any $\mathcal{T}$. Alternatively, the rules for m can be modified in a similar way to make the program congruent for m. Then the program is also congruent for w (without modifying the rule for w).

The program transformations above can be seen as usual CLP programming tricks.

For congruent hybrid programs, the problem of two equalities does not exist (also it does not exist for external theories without equality). Example 3.6 informally introduces programming techniques for constructing congruent programs. Now we present two simple criteria assuring that a program is congruent (congruency is undecidable, like other non-trivial semantic properties of programs). The first criterion refers to the free equality theory (CET, Clark equality theory) [6].

Definition 3.7 CET (*Clark equality theory*) consists of equality axioms

$$\begin{array}{ll} x = x, & \\ \overline{x} = \overline{y} \rightarrow f(\overline{x}) = f(\overline{y}) & \text{for each } f \in \mathcal{F}, \\ \overline{x} = \overline{y} \rightarrow (p(\overline{x}) \rightarrow p(\overline{y})) & \text{for each predicate symbol } p, \text{ including } =, \end{array}$$

and freeness axioms

$$\begin{array}{ll} f(\overline{x}) = f(\overline{y}) \rightarrow \overline{x} = \overline{y} & \text{for each } f \in \mathcal{F}, \\ f(\overline{x}) \neq g(\overline{y}) & \text{for each pair of distinct } f, g \in \mathcal{F}, \\ x \neq t & \text{for each non variable term } t \text{ such that} \\ & \text{the variable } x \text{ occurs in } t. \end{array}$$

If the set $\mathcal{F}$ of function symbols is finite then CET additionally contains the weak domain closure axiom WDCA:[9]

$$\bigvee_{f \in \mathcal{F}} \exists \overline{y}\, (x = f(\overline{y})).$$

The syntactic equality over the Herbrand universe is a model of CET. When $\mathcal{F}$ contains only constants then CET reduces to the unique name assumption (UNA).

If the equality $='$ of $\mathcal{T}$ satisfies CET then each program $(P, \mathcal{T})$ is congruent (as then if t, u are ground terms then $t =' u$ implies that the terms are identical). Apparently, for this reason, some approaches of combining rules and ontologies require that the ontology satisfies the unique name assumption.

Another sufficient criterion is syntactic. Program $(P, \mathcal{T})$ is congruent if in each rule $H \leftarrow C, L_1, \ldots, L_n$ of P all the arguments $t_1, \ldots, t_n$ of the head $H = p(t_1, \ldots, t_n)$ are variables, and any variable occurs at most once in $H, L_1, \ldots, L_n$ (thus the remaining occurrences of the variable are in the constraint C of the rule). The proof that such $(P, \mathcal{T})$ is congruent is based on the fact that for any model M of $\mathcal{T}$ if $\mathcal{T} \models t_1 =' u_1, \ldots, t_n =' u_n$ (for ground terms $t_1, \ldots, t_n, u_1, \ldots, u_n$) then a rule $p(t_1, \ldots, t_n) \leftarrow \overline{B}$ is in P/M iff $p(u_1, \ldots, u_n) \leftarrow \overline{B}$ is in P/M.

As an example, notice that the rule for w in P'' from Example 3.6 satisfies the sufficient condition, and the rules for m do not. Notice also that the condition is different from usually considered safeness conditions (roughly speaking, the former forbids, and the latter require certain variable occurrences).

[9] This axiom is needed for CET to be complete, in the sense that any closed formula (with $=$ as its only predicate symbol) has the same logical value in each model of CET. Consider for instance $\mathcal{F} = \{a\}$ and $\exists x (x \neq a)$. This formula is true in some models of CET without WDCA, but false in its Herbrand models. (If $=$ is the only predicate symbol then the Herbrand model is unique).

It is rather obvious how to construct programs satisfying this syntactic restriction, provided that the set of constraints includes equalities $t =' u$ of $\mathcal{T}$. Instead of placing a non-variable term t as an argument of the head of a rule, use a new variable x_t and add $x_t =' t$ to the constraint of the rule. Instead of writing more than one occurrences of a variable x in the rule literals of a rule, replace each (but one) occurrence of x by a new distinct variable x' and add $x' =' x$ to the constraint of the rule.

4 Reasoning with hybrid rules

Now we present a way of computing the well-founded semantics of Definition 3.2. Like in logic programming, the task is to find instances of a given goal formula G which are true in the well-founded semantics of a given program. Similarly to logic programming, our operational semantics is defined in terms of search trees. After introducing the operational semantics, we prove its soundness and completeness, the latter for a restricted class of programs.

4.1 Constraints for the operational semantics

To construct the operational semantics we impose certain requirements on the external theory and the set of constraints. We need to deal explicitly with the syntactic equality $=$ and its negation. So we require that $=$ is a constraint predicate symbol and the external theory $\mathcal{T}$ includes the axioms CET (cf. Sect. 3.3). An external theory $\mathcal{T}'$ which does not satisfy this condition can be easily converted to a $\mathcal{T}$ which does ($\mathcal{T}'$ may be a theory without equality, or contain equality $='$ not satisfying CET). Namely $\mathcal{T}$ is the union $\mathcal{T} = \mathcal{T}' \cup \text{CET}$. Reasoning in such $\mathcal{T}$ can be implemented employing Prolog and a reasoner for $\mathcal{T}'$ [14,15]. The former deals with $=$, the latter with the predicates of $\mathcal{T}'$.

The operational semantics constructs new constraints using conjunction, disjunction, negation, and existential quantification. So we require that the set of constraints is closed under these operations. This imposes restrictions on the constraints. For instance many DLs do not allow negation of roles; for such DL a formula of the form $r(X, Y)$ cannot be a constraint. The actual choice of constraints is outside of the scope of this paper. It depends on the chosen external theory and the available reasoner for it. For instance, if a formula C is a constraint without $=$ then the reasoner should be able to check whether C is satisfiable in $\mathcal{T}'$ (where $\mathcal{T}$ and $\mathcal{T}'$ are as above).

4.2 Operational semantics

The operational semantics presented below is a generalization of SLS-resolution [34], which is extended by handling constraints originating from the hybrid rules. It is based on the constructive negation approach presented in [11,12]. In logic programming, the term constructive negation stands for generalizations of negation as failure (NAF) (see e.g. [3]). NAF provides a way of checking whether a given negative goal is a consequence of the program (under a relevant semantics). Constructive negation, roughly speaking, finds instances of a negative goal which are consequences. The main contribution of the operational semantics presented here is dealing with hybrid programs and arbitrary external theories. The constructive negation method of Refs. [11,12] dealt with logic programs, the equality was the only constraint predicate and CET was the constraint theory.

The operational semantics is similar to SLDNF- and SLS-resolution [27,34]. For an input goal, a derivation tree is constructed; its nodes are goals. Whenever a negative literal is selected in some node, a subsidiary derivation tree is constructed. So a tree of trees is obtained.

Definition 4.1 By the **restriction** $F|_V$ of a formula F to a set V of variables we mean the formula $\exists x_1, \ldots, x_n F$ where $x_1, \ldots, x_n$ are those free variables of F that are not in V. By $F|_{F'}$ we mean $F|_V$, where V are the free variables of formula F'.

By a **goal** we mean a conjunction of the form $C, L_1, \ldots, L_n$ $(n \geq 0)$, where each L_i is a rule literal and C is a constraint (the **constraint** of the goal). Consider a goal $G = C, \overline{L}, p(\overline{t}), \overline{L'}$ and a rule $R = p(\overline{u}) \leftarrow C', \overline{K}$, such that no variable occurs both in G and R. We say that the goal

$$G' \;=\; \overline{t} = \overline{u}, C, C', \overline{L}, \overline{K}, \overline{L'}$$

is **derived** from G by R, with the selected atom $p(\overline{t})$, if the constraint $\overline{t} = \overline{u}, C, C'$ is satisfiable.

We inductively define two kinds of derivation trees: t-trees and tu-trees. Their role is to find out when a given goal is **t**, or respectively when it is **t** or **u**. Informally, if a constraint C is a leaf of a t-tree with the root G then C implies that G is **t** in the well-founded semantics of the program (more generally, the same holds if C is a disjunction of such leaves). On the other hand, for a tu-tree we define a notion of its *cross-section*. If $C_1, \ldots, C_n$ are the constraints of the goals of a cross-section of a tu-tree with the root G then, roughly speaking, $\neg(C_1 \vee \cdots \vee C_n)$ implies that G is **f** in the well-founded semantics of the program. A formal explanation is provided by the soundness Theorem 4.9 in the next section.

For correctness of the definition (to avoid circularity), we assign ranks to the trees. This is a standard technique employed in similar definitions [27,12,34]. In the general case ranks are countable ordinals, but natural numbers are sufficient for a language where the function symbols are constants. The children of nodes with an atom selected are defined as in the standard SLD-resolution. The only difference is that instead of explicit unification we employ equality constraints. The children of nodes with a negative literal selected are constructed employing the results of tu- (t-) trees of lower rank. A t-tree refers to tu-trees and vice versa. This is basically a reformulation of the corresponding definitions of Ref. [11,12].

Definition 4.2 (*Operational semantics*) A t-tree (tu-tree) of rank $k \geq 0$ for a goal G w.r.t. a program $(P, \mathcal{T})$ satisfies the following conditions. The nodes of the tree are (labeled by) goals. In each node a rule literal is selected, if such a literal exists. A node containing no rule literal is called **successful**, a branch of the tree with a successful leaf is also called **successful**.

1. A constraint $(C_1 \vee \cdots \vee C_n)|_G$ $(n \geq 0)$[10] is an **answer** of the t-tree if $C_1, \ldots, C_n$ are (some of the) successful leaves of the t-tree (it is not required that all the successful leaves are taken).
2. By a **cross-section** (or frontier) of a tu-tree we mean a set F of tree nodes such that each successful branch of the tree has a node in F. Let F be a cross-section of the tu-tree and $CF = \{ C_1, \ldots \}$ the constraints of the nodes in F.
   If $CF = \{ C_1, \ldots, C_n \}$ is finite then the constraint $\neg(C_1|_G), \ldots, \neg(C_n|_G)$ (the negation of $\bigvee(C_i|_G)$) is called a **negative answer** of the tu-tree.
   If CF is infinite then a constraint C which implies $\neg(C_i|_G)$ for each $C_i \in CF$ is called a **negative answer** of the tu-tree. Moreover, it is required that each free variable of C is a free variable of G.

[10] If $n = 0$ then by $C_1 \vee \cdots \vee C_n$ we mean **false**, and by $C_1 \wedge \cdots \wedge C_n$ we mean **true**.

3. If (in the t-tree or tu-tree) the selected literal A in a node G' is an atom then, for each rule R of P, a goal derived from G' with A selected by a variant R' of R is a child of G', provided such a goal exists. Moreover, it is required that no variable in R' occurs in the tree on the path from the root to G'.
4. Consider a node $G' = C, \overline{L}, \neg A, \overline{L'}$ of the t-tree (tu-tree), in which the selected literal $\neg A$ is negative. The node is a leaf or has one child, under the following conditions.
   (a) If the tree is a t-tree then
      i. G' is a leaf, or
      ii. G' has a child $C', C, \overline{L}, \overline{L'}$, where C' is a negative answer of a tu-tree for C, A of rank $< k$, and C', C is satisfiable.
   (b) If the tree is a tu-tree then
      i. G' has a child $C, \overline{L}, \overline{L'}$, or
      ii. G' has a child $C', C, \overline{L}, \overline{L'}$, where $C' = \neg C''$ is the negation of an answer C'' of a t-tree for C, A of rank $< k$, and C', C is satisfiable, or
      iii. G' is a leaf and there exists an answer C'' of a t-tree for C, A of rank $< k$ such that $\neg C'', C$ is unsatisfiable.

An informal explanation for case 2 is that the constraints of the cross-section include all the cases in which G is **t** or **u**, thus their negation implies that G is **f**. A useful intuition is that adding a negative answer C to the nodes of the tu-tree results in a failed tree—a tree for C, G without any successful leaf. (Whenever C_i is the constraint of a node of the cross-section, the constraint C, C_i is unsatisfiable. The same holds for any node which is a descendant of some node of the cross-section).

An informal explanation for case 4 is that in a t-tree (case 4(a)ii) C' implies that C, A is **f**, equivalently $\neg(C, A)$ is **t**. Hence C', C implies that $\neg A$ is **t**. In a tu-tree (4(b)ii) $\neg C''$ includes all the cases in which C, A is not **t**. Hence $\neg C'', C$—the constraint of the child—includes all the cases in which A is not **t**, equivalently in which $\neg A$ is **t** or **u**.

Notice that in case 4(a)i the node $G' = C, \overline{L}, \neg A, \overline{L'}$ may unconditionally be a leaf of a t-tree (of any rank). This corresponds to the fact that $C' = \neg C$ is a negative answer for any tu-tree for C, A. (Take the cross-section $\{C, A\}$). Hence in the supposed child of G' (case 4(a)ii) the constraint $\neg C, C$ is unsatisfiable. Conversely, according to 4(b)i, node $G' = C, \overline{L}, \neg A, \overline{L'}$ in a tu-tree may have $C, \overline{L}, \overline{L'}$ as the child. This corresponds to the fact that $C'' =$ **false** is an answer of any t-tree. Hence C is equivalent to $\neg C'', C$ (which is the constraint obtained in 4(b)ii). Thus 4(b)i is a special case of 4(b)ii.

Example 4.3 Consider a query $w(c)$ for the hybrid program of Example 2.3. It can be answered by the operational semantics by construction of the following trees (sometimes we replace a constraint by an equivalent one).

1. A t-tree for $w(c)$:

$$
\begin{array}{ccc}
 & w(c) & \\
 & | & \\
 & X=c, m(X,Y), \neg w(Y) & \\
 & / \quad \backslash & \\
X=c, Y=f, \neg Fi(f), \neg w(Y) & & X=c, Y=d, \neg w(Y) \\
| & & | \\
X=c, Y=f, \neg Fi(f) & & X=c, Y=d, \neg(X=c, Y=d, \neg E(f))
\end{array}
$$

   The tree refers to negative answers derived in the cases 2, 4 below. The constraint in the second leaf is equivalent to $X=c, Y=d, E(f)$ (as $\alpha \wedge \neg(\alpha \wedge \beta)$ is equivalent to

$\alpha \wedge \neg\beta$). The answer obtained from the two leaves C_1, C_2 of the tree is $\exists X, Y\ (C_1 \vee C_2)$. It is equivalent to $\neg Fi(f) \vee E(f)$. As this constraint is a logical consequence of the ontology, $w(c)$ holds in each well-founded model of the program.

2. A tu-tree for $X = c, Y = d, w(Y)$, employing an answer from the t-tree from case 3:

$$\begin{array}{c} X = c, Y = d, w(Y) \\ | \\ X = c, Y = d, X' = Y, m(X', Y'), \neg w(Y') \\ | \\ X = c, Y = d, X' = d, Y' = e, \neg w(Y') \\ | \\ X = c, Y = d, X' = d, Y' = e, \neg(X = c, Y = d, X' = d, Y' = e, E(f)) \end{array}$$

   The leaf is equivalent to $X = c, Y = d, X' = d, Y' = e, \neg E(f)$, see the explanation in the previous case. Hence from the cross-section containing the leaf we obtain a negative answer equivalent to $\neg(X = c, Y = d, \neg E(f))$ and to $D = \neg(X = c, Y = d) \vee E(f)$. Informally, D implies falsity of the root of the tu-tree. (For a formal treatment see Theorem 4.9 and Lemma 4.8 below). Hence $E(f)$ implies $\neg w(d)$. Formally, if $M_0 \models E(f)$ for some model M_0 of $\mathcal{T}$ then $w(d)$ is false in $WF(P/M_0)$ (the well-founded model of P based on M_0).

3. A t-tree for $Y' = e, w(Y')$ employing a negative answer from case 4:

$$\begin{array}{c} Y' = e, w(Y') \\ | \\ Y' = e, X'' = Y', m(X'', Y''), \neg w(Y'') \\ | \\ Y' = e, X'' = Y', X'' = e, Y'' = f, E(f), \neg w(Y'') \\ | \\ Y' = e, X'' = e, Y'' = f, E(f) \end{array}$$

   The corresponding answer is (equivalent to) $Y' = e,\ E(f)$. Informally, the answer implies $Y' = e, w(Y')$. From Lemma 4.8 below it follows that if $E(f)$ holds in some model M_0 of $\mathcal{T}$ then $w(e)$ is true in the corresponding well-founded model of P.
   Notice that if $C, Y' = e$ is a satisfiable constraint then C may be added to the nodes of the tree (maybe with renaming of variables X'', Y''). Hence $C, Y' = e, E(f)$ is an answer for $C, Y' = e, w(Y')$. To construct the t-tree of case 2 we use $C = (X = c, Y = d, X' = d)$.

4. A tu-tree for $Y = f, w(Y)$, with atom $m(X', Y')$ selected in the leaf:

$$\begin{array}{c} Y = f, w(Y) \\ | \\ Y = f, Y = X', m(X', Y'), \neg w(Y') \end{array}$$

   From the empty cross-section a negative answer **true** is obtained. So $w(f)$ is false in the well-founded semantics of the program. Similarly, **true** is a negative answer for $C,\ Y = f,\ w(Y)$, where C is an arbitrary constraint.

Various simplifications of t- (tu-) trees are possible. For instance in case 3 of the last example the nodes of the tree may be replaced by $w(e);\ m(e, Y), \neg w(Y);\ E(f), \neg w(f);\ E(f)$ [14, Example 2]. This issue is outside of the scope of this paper.

We do not deal here with actual implementing of the operational semantics (an implementation is described in [14]). We only mention that—similarly as in CLP—it is not necessary

to check satisfiability of the constraint for each node. The answers (negative answers) of trees obtained in this way are logically equivalent to those of t-trees (tu-trees) from Definition 4.2.

4.3 Soundness

In this section, we prove soundness of the operational semantics of hybrid programs (Definition 4.2) with respect to their declarative semantics (Definition 3.2). Before the actual proof we discuss ground instances of goals and trees, and introduce safe programs and goals. These notions are employed in the proof.

For our proofs, we use the characterization of the well-founded semantics of logic programs from Sect. 2.1. So for a given model M_0 of the external theory, the well-founded model of the program is $\Psi^{\alpha}_{P/M_0}(\emptyset)$ for some α.

4.3.1 Ground instances of trees

By an **extension** of a substitution θ we mean any substitution of the form $\theta \cup \theta'$ (where $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ and $\theta' = \{y_1/u_1, \ldots, y_m/u_m\}$ are substitutions with disjoint domains, $\{x_1, \ldots, x_n\} \cap \{y_1, \ldots, y_m\} = \emptyset$).

By a **grounding substitution** for the variables of a formula F (or just "for F") we mean a substitution replacing the free variables of F by ground terms (the domain of the substitution may include other variables).

Let $G = C, \overline{L}$ be a goal and M_0 a model of $\mathcal{T}$. Let θ be a grounding substitution for the variables of G (notice that $C\theta$ has no free variables). If $C\theta$ is true in M_0 then we say that θ is **applicable** to G (w.r.t. M_0), and by the result $G\theta$ of applying θ to G we mean the ground goal $\overline{L}\theta$; it is called a **ground instance** of G. Similarly, we say that θ is *applicable* to a rule $H \leftarrow C, \overline{L}$; the result $H\theta \leftarrow \overline{L}\theta$ is called a **normal ground instance** of the rule.[11] Notice that a rule $R\theta$ is a normal ground instance of a rule $R \in P$ w.r.t. M_0 iff $R\theta \in (P/M_0)$.

Consider a t-tree or tu-tree Tr for G and θ as above. A **ground instance** $Tr\theta$ of Tr w.r.t. M_0 is defined recursively as follows. The nodes of $Tr\theta$ are ground instances of (some) nodes of Tr, each node H of $Tr\theta$ **corresponds** to a node G' of Tr such that H is a ground instance of G'. The root of $Tr\theta$ is $G\theta$ and it corresponds to the root G of Tr. If a node $G'\theta'$ of $Tr\theta$ corresponds to node G' of Tr (where θ' is a grounding substitution for the variables of G'), G'' is a child of G' in Tr, θ'' is an extension of θ' then $G''\theta''$ is a child of $G'\theta'$ in $Tr\theta$, provided that θ'' is applicable to G''.

A node of $Tr\theta$ corresponding to a successful leaf of Tr will be called a **successful** leaf of $Tr\theta$.

Example 4.4 Consider a program $P = \{p(a) \leftarrow q(y)\}$. The t-tree (or tu-tree) Tr for $p(x)$ consists of two nodes; the child of $p(x)$ is $x = a, q(y)$. For $\theta = \{x/b\}$ the ground instance $Tr\theta$ consists of one node $p(b)$. For $\sigma = \{x/a\}$ the root $p(a)$ of $Tr\sigma$ has a child $q(t)$ for each ground term t. Each node $q(t)$ in $Tr\sigma$ corresponds to the node $x = a, q(y)$ of Tr.

Notice that if $G'\theta'$ and its child $G''\theta''$ in $Tr\theta$ correspond, respectively, to G' and G'' in Tr, and G'' is derived from G' by a rule $R \in P$ then $G''\theta''$ is derived from $G'\theta'$ by a normal ground instance $R\sigma \in (P/M_0)$ of R (this means that $R\sigma$ is $H \leftarrow \overline{B}$, H is an atom in $G'\theta'$ and G'' is $G'\theta'$ with H replaced by $\overline{B}$). Thus if a leaf $G'\theta'$ of $Tr\theta$ corresponds to a node G'

[11] Definition 3.2 employs another kind of ground instance, namely $H\theta \leftarrow C\theta, \overline{L}\theta$. As ground instances in that sense are not used below, we sometimes skip the word "normal." To simplify notation, we write $R\theta$ for a normal ground instance of a rule R, when this does not lead to ambiguity.

of Tr, and a positive literal A is selected in G' then no normal ground instance of a rule in the program has the head $A\theta$. If no negative literal is selected in Tr then $Tr\theta$ is an SLD-tree ([27]) for program (P/M_0).

4.3.2 Safeness

We introduce a notion similar to DL-safeness [32,36], but taking into account that constraints may contain equality $=$.

Definition 4.5 Let C be a constraint. A variable x is **bound** in C to a ground term t (to a variable y) if $\mathcal{T} \models C \rightarrow x = t$ (respectively $\mathcal{T} \models C \rightarrow x = y$).

For instance, in $x = f(y),\ y = a$ variable x is bound to ground term $f(a)$, and y is bound to a. Notice that any variable is bound to itself independently of the constraint. A simple sufficient condition is that, for variables $x_0, \ldots, x_n$ $(n \geq 0)$ and a ground term t, if C is a conjunction of constraints $C_1, \ldots, C_l$ and the set $\{C_1, \ldots, C_l\}$ contains equalities $x_0 = x_1, \ldots, x_{n-1} = x_n$ (resp. $x_0 = x_1, \ldots, x_{n-1} = x_n, x_n = t$) then x_0 is bound to x_n (resp. to t) in C.

Definition 4.6 A rule $R = H \leftarrow C, \overline{L}$, where C is the constraint of R, is **safe** if

- each variable of H,
- each variable of a negative literal of $\overline{L}$, and
- each free variable of C

is bound in C to a ground term or to a variable appearing in a positive literal in $\overline{L}$.

Let V be a set of variables. When the conditions above are satisfied with possible exception for the variables from V then we say that R is **safe apart** of V.

A set of rules is *safe* if all its rules are safe. A hybrid program $(P, \mathcal{T})$ is *safe* if P is safe. A goal $G = C, \overline{L}$ is *safe* if the rule $p \leftarrow G$ is safe (where p is a 0-argument predicate symbol). G is *safe apart* of V if the rule $p \leftarrow G$ is safe apart of V.

If the root of a t-tree (tu-tree) for a safe program is safe then any node of the tree is safe. Hence, in the constraint of a successful leaf, all the free variables are bound to ground terms. In the Appendix we prove a more general property:

Lemma 4.7 *Let $(P, \mathcal{T})$ be a safe program, V a set of variables, and G a goal. Consider a t-tree (or a tu-tree) with the root G.*

1. *Each node of the tree is safe apart of V_0, where V_0 is the set of free variables of G.*
2. *Assume that no variable from V occurs in any variant of a rule from P used in constructing the tree. If G is safe apart of V then each node of the tree is safe apart of V.*

4.3.3 Soundness theorem

We will say that the set of constraints has the **witness property** if for any model M of the external theory $\mathcal{T}$ and for any constraint C, whenever $M \models \exists C$ then $M \models C\theta$ for some grounding substitution θ for C. The witness property is implied by the parameter names assumption (PNA) [8] that restricts the interpretations of $\mathcal{T}$ to those in which every domain element is a value of a ground term.

The operational semantics may be not sound for constraints without witness property and non-safe programs. As an example take $P = \{\, p \leftarrow q(x) \,\}$ and $\mathcal{T} = \{\exists x.q(x)\}$. Then (a constraint $\exists x.q(x)$ equivalent to) **true** is an answer of the t-tree for p. However $(P, \mathcal{T}) \not\models_{\mathrm{wf}} p$ (as there exist models of $\mathcal{T}$ in which every ground instance of $q(x)$ is false).

Lemma 4.8 (Soundness) *Consider a program $(P, \mathcal{T})$, a goal G, a model M_0 of $\mathcal{T}$, and a countable ordinal number k. Assume that M_0 is a Herbrand interpretation, or the set of constraints has the witness property, or P is safe.*

1. *If C is an answer of a t-tree of rank k for G then for any grounding substitution θ (for the variables of G) $M_0 \models C\theta$ implies $\Psi^{k+1}_{P/M_0}(\emptyset) \models_3 G\theta$.*
2. *If C is a negative answer of a tu-tree of rank k for G then for any grounding substitution θ (for the variables of G) $M_0 \models C\theta$ implies $\Psi^{k+1}_{P/M_0}(\emptyset) \models_3 \neg G\theta$.*

The proof is presented in the Appendix.[12] It is based on studying ground instances of the t- (tu-) tree for G and viewing them as SLD-trees of a program $(P/M_0)/_t I$ (respectively a program related to $(P/M_0)/_{tu} I$) from the definition of Ψ_{P/M_0}.

As a corollary we obtain:

Theorem 4.9 (Soundness) *Let $(P, \mathcal{T})$ be a hybrid program and $G = C_0, \overline{L}$ a goal (where C_0 is the constraint of G). Assume that P is safe, or the set of constraints has the witness property.*

If C is an answer of a t-tree for $(P, \mathcal{T})$ and G then, for any substitution θ, $\mathcal{T} \models C\theta$ implies $(P, \mathcal{T}) \models_{\mathrm{wf}} \overline{L}\theta$.

If C is a negative answer of a tu-tree for $(P, \mathcal{T})$ and G then, for any substitution θ, $\mathcal{T} \models C\theta$ implies $(P, \mathcal{T}) \models_{\mathrm{wf}} \neg\overline{L}\theta$.

Proof Let $G = C_0, \overline{L}$ and C be an answer of the t-tree. $\mathcal{T} \models C\theta$ implies $M_0 \models C\theta\theta'$ for any substitution θ' and any model M_0 of $\mathcal{T}$. Consider a θ' such that $x\theta'$ is ground for each free variable x of $(C, \overline{L})\theta$. From Lemma 4.8, applied to M_0, C, G, and the substitution $\theta\theta'$, it follows that $M \models_3 G\theta\theta'$, where M is the well-founded model or P based on M_0. Notice that $G\theta\theta' = \overline{L}\theta\theta'$. As $M \models_3 \overline{L}\theta\theta'$ for each θ' as above and M is a Herbrand interpretation, we have $M \models_3 \overline{L}\theta$. As the latter holds for each well-founded model of P, we obtain $(P, \mathcal{T}) \models_{\mathrm{wf}} \overline{L}\theta$.

The proof for a negative answer of a tu-tree is analogical. □

It may be desirable to have an operational semantics which is sound also for non-safe programs and constraints without the witness property. This can be obtained by employing, in formula restrictions (Definition 4.1), a certain non-standard quantifier $\exists'$ instead of $\exists$. For the new quantifier it holds that if $I \models \exists' F$ then $I \models F\theta$ for some grounding substitution θ for F (for any formula F and interpretation I). The details are outside the scope of this paper.

4.4 Completeness

In a general case, our operational semantics is not complete. Roughly speaking, the reason is using only finite constraint formulae as (negative) answers[13] in case 4 of Definition 4.2. We show completeness of our operational semantics for the case where the Herbrand universe is finite and the program and goals are safe. The completeness result includes independence from the selection rule.

[12] From the proof it follows that safeness is needed only for the rules that have been used in constructing t-trees (the t-trees referred to, directly or indirectly, by the t- (tu-) tree for G). Alternatively, the witness property is necessary only for the constraints that are successful leaves of these t-trees.

[13] In Refs. [11,12] this problem was solved by allowing an infinite set of children of a node with a negative literal selected. Example 4.10 in [11] shows that this is actually necessary. Thus it provides a counterexample for completeness of the operational semantics of Definition 4.2.

We first present a technical lemma about simplifying goals for which t-trees (tu-trees) are constructed. Then we restrict our considerations to safe programs over a finite universe, show how a kind of a most general (negative) answer for a given tree can be obtained, and define a notion of a maximal t- (tu-) tree. Intuitively, a maximal tree (of a sufficiently high rank) derives everything that is required by the declarative semantics. This is made formal in a completeness lemma, from which completeness of the operational semantics follows.

The following lemma shows how a tree for A may replace a tree for C, A. So, for a fixed A, many trees for goals C, A may be replaced by a single tree. On the other hand, for a fixed C the tree for A may have more nodes than the corresponding one for C, A.

Lemma 4.10 *Consider a program $(P, \mathcal{T})$. Let A be an atom and C a constraint. If $\neg(C_1|_A), \ldots, \neg(C_n|_A)$ is a negative answer of a tu-tree (respectively a negation of an answer of a t-tree) of rank k for A then $\neg\left(C, (C_1|_A)\right), \ldots, \neg\left(C, (C_n|_A)\right)$, or equivalently $\neg C \vee \neg(C_1|_A), \ldots, \neg(C_n|_A)$, is a negative answer of a tu-tree (the negation of an answer of a t-tree) of rank k for C, A.*

Proof Without loss of generality we can assume that if a variable x occurs both in the tree for A and in C then x occurs in A. If $C', \overline{L}$ is a node of the tree for A then $C, C', \overline{L}$ is a node of the tree for C, A provided that C, C' is satisfiable. Assume the negative answer for A is obtained from a finite cross-section. Consider a "corresponding" cross-section of the tree for C, A whose node constraints are those of C, C_i that are satisfiable. The corresponding negative answer is $\neg((C, C_1)|_A), \ldots, \neg((C, C_n)|_A)$. Each $(C, C_i)|_{C,A}$ is equivalent to $C, (C_i|_A)$. The cases of an infinite cross-section, and of a t-tree, are similar. □

4.4.1 Maximal trees

For this section, we assume that the Herbrand universe is finite. Hence the alphabet of function symbols is finite and contains only constants.

A t- (tu-) tree may be infinite. Moreover, the set of nodes in the tree with a negative literal selected may be infinite. Hence the tree may refer to an infinite set of subsidiary trees. The answers (negative answers) of the tree may be obtained from an infinite set of successful leaves (an infinite cross-section); so it seems that we have to deal with an infinite set of answers (as there may not exists one which implies all the others). In what follows we show how to avoid infinite sets of answers.

Notice first that the set of selected negative literals in a t- (tu-) tree is finite, up to renaming of variables (as the Herbrand universe is finite). By Lemma 4.10, instead of constructing a possibly infinite set of subsidiary tu- (t-) trees for goals of the form C, A, it is sufficient to construct a finite set of tu- (t-) trees for goals of the form A.

In a t- (tu-) tree with a safe root, if C is a successful leaf then each free variable x of C is bound to a constant c_x. This defines a grounding substitution $\theta = \{x_1/c_{x_1}, \ldots, x_m/c_{x_m}\}$ for the free variables $x_1, \ldots, x_m$ of C. Now C is equivalent to $x_1 = c_{x_1}, \ldots, x_m = c_{x_m}, C$ and to $x_1 = c_{x_1}, \ldots, x_m = c_{x_m}, C\theta$.

By a **grounded constraint** from a program P (from a goal G) we mean a constraint $C\sigma$ such that constraint C is the constraint of a rule of P (the constraint of G) and the substitution σ replaces all the free variables of C by constants. A constraint is in a **solved form** for a program P and an initial goal G if it is a conjunction of constraints of the form $x = c$, C_0 or $\neg C_0$, where x is a variable, c is a constant, and C_0 is a grounded constraint from P or G.

The set of grounded constraints from a given program or a given goal is finite. So the set of constrains in solved form (for P and G) with the free variables from G is finite, up to equivalence.

We now show that, under certain conditions, the successful leaves of t- (tu-) trees may be seen as disjunctions of constraints in solved form.

Lemma 4.11 *Let the Herbrand universe be finite and Tr be a t-tree (tu-tree) for a safe goal G and a safe program $(P, \mathcal{T})$. Assume that each negative answer (negation of an answer) employed in the tree is of the form $\neg C_1, \ldots, \neg C_n$, where each C_i is in solved form for P and G. Then the constraint of any success leaf C of the tree is equivalent to a disjunction of constraints in solved form. Also, $C|_G$ is equivalent to a disjunction of constraints in solved form for P and G.*

Proof Consider a success leaf C. Each free variable x of C is bound in C to a constant c_x. Let $\theta = \{x_1/c_{x_1}, \ldots, x_m/c_{x_m}\}$ (where $x_1, \ldots, x_m$ are the free variables of C). C is equivalent to $x_1 = c_{x_1}, \ldots, x_m = c_{x_m}, C$ and to $x_1 = c_{x_1}, \ldots, x_m = c_{x_m}, C\theta$ ($C\theta$ has no free variables).

$C\theta = C_1, \ldots, C_n$, where each C_i is a ground equality, or a grounded constraint from P or from G, or a $\neg C_i'\theta$ where C_i' is a constraint in a solved form. $\neg C_i'$ is equivalent to $\neg D_1 \vee \cdots \vee \neg D_l$, where each D_i is of the form $x = c$ or is a possibly negated grounded constraint from P or from G. Thus $\neg C_i'\theta$ is equivalent to **true** (if some ground disequality $\neg D_i\theta$ is true), or to $\neg D_{j_1} \vee \cdots \vee \neg D_{j_{l'}}$, where each D_{j_i} is a possibly negated grounded constraint from P or G. Each C_i which is a ground equality is equivalent to **true** (as it is satisfiable).

So applying distributivity $(\phi, (\psi \vee \psi') \equiv \phi, \psi \vee \phi, \psi')$ to $x_1 = c_{x_1}, \ldots, x_m = c_{x_m}$, $C\theta$ results in an equivalent disjunction of constraints in solved form. Removing from the latter constraint each $x_i = c_i$ where x_i not free in G produces a disjunction of constraints in solved form equivalent to $C|_G$. □

As discussed earlier, the set of disjunctions of constraints in solved form (for P and G) with the free variables from G is finite, up to equivalence. Thus from the lemma it follows that the set of success leaves of the t-tree (tu-tree) is equivalent to a finite set of disjunctions of constraints in solved form. Formally: If the tree satisfies the conditions of Lemma 4.11 then there exists a constraint D (we will call it a **finite answer** of the t-tree, resp. **finite pseudo-answer** of the tu-tree), such that

$$M_0 \models C|_G\theta \text{ for some success leaf } C \text{ of the tree} \quad \text{iff} \quad M_0 \models D\theta,$$

for every model M_0 of $\mathcal{T}$ and every substitution θ grounding the variables of G. Moreover, D is a disjunction of constraints in solved form, for P and the root G of the tree (D is equivalent to a constraint $(C_1|_G) \vee \cdots \vee (C_m|_G)$ where each C_i is a success leaf of the tree).

Notice that the negation of a pseudo-answer of a tu-tree is a negative answer of the tree (the corresponding cross-section contains all the successful leaves). Informally, a finite answer is a most general answer that can be obtained from a given t-tree, and the negation of a finite pseudo-answer is a most general negative answer that can be obtained from a given tu-tree.

Now for a given rank and goal, we define a maximal t- (tu-) tree. The intention is that the tree provides a (negative) answer which is more general than other (negative) answers for this goal of the same rank.

Definition 4.12 Let the Herbrand universe be finite, and $(P, \mathcal{T})$ be a safe program. A **maximal** t-tree (tu-tree) for a goal and $(P, \mathcal{T})$ is defined inductively:

A maximal t-tree of rank 0 is a t-tree in which each node with a negative literal selected is a leaf.

A maximal tu-tree of rank 0 is a tu-tree in which if a negative literal is selected in a node G' and the constraint of G' is C then the constraint of the child of G' is C (cf. Definition 4.2 case 4(b)i).

A maximal t-tree (tu-tree) of rank $k > 0$ is a t-tree (tu-tree) in which a node G' with a negative literal $\neg A$ selected has a child G'' iff $C, \neg D$ is satisfiable, where C is the constraint of G' and D is a finite pseudo-answer (a finite answer) of a maximal tu-tree (t-tree) for A of rank $k-1$; moreover, $C, \neg D$ is the constraint of G''.[14]

The definition is correct: From Lemmas 4.10 and 4.11 by induction on the rank we obtain that, for a safe program P and a safe goal G, the tree described in the definition satisfies the conditions of Lemma 4.11, it has a finite (pseudo-) answer which is a disjunction of constraints in solved form;[15] hence the finite (pseudo-) answers employed in the definition exist. Notice that a maximal t- (tu-) tree is defined for any safe program and any goal G_0. If G_0 is safe then the tree has a finite (pseudo) answer.

4.4.2 Completeness theorem

By a **selection rule** we mean a function which, given a sequence of goals $G_0, \ldots, G_n$, selects a rule literal in G_n provided G_n contains at least one rule literal and $G_0, \ldots, G_n$ is a prefix of a branch of a t-, tu-, or SLD-tree.

Now we are ready to state and prove completeness of the operational semantics. We first show that a (negative) answer of any maximal (tu-) t-tree for G of rank j—speaking informally—describes all the instances of G that are true (false) in a corresponding approximation $\Psi^k_{P/M_0}(\emptyset)$ of the well-founded model of P/M_0.

Lemma 4.13 (Completeness) *Assume that the Herbrand universe is finite. Consider a safe program $(P, \mathcal{T})$. Let G be a safe goal, C_0 be the constraint of G, and $k > 0$ be a natural number. Consider a selection rule $\mathcal{R}$, and a maximal t-tree and a maximal tu-tree for G of rank $k-1$ if G does not contain a negative literal, and of rank k otherwise.*

Let M_0 be a model of $\mathcal{T}$, and θ be a grounding substitution for the variables of G such that $M_0 \models C_0\theta$

1. *If $\Psi^k_{P/M_0}(\emptyset) \models_3 G\theta$ and C is a finite answer of the t-tree then $M_0 \models C\theta$.*
2. *If $\Psi^k_{P/M_0}(\emptyset) \models_3 \neg G\theta$ and D is a finite pseudo-answer of the tu-tree (and thus $\neg D$ is a negative answer) then $M_0 \models \neg D\theta$.*

The basic idea of the proof is that in case 1. it follows that there exists a successful SLD-derivation for $G\theta$ and a certain program $(P/M_0)/_t J$. This derivation is shown to be a branch of a ground instance of the maximal t-tree for G. In case 2. there does not exist a successful SLD-derivation for $G\theta$ and a certain program $(P/M_0)/_{tu} J$. However, such a derivation is shown to be a branch of a ground instance of the maximal tu-tree for G, under assumption that $M_0 \models D\theta$. Hence $M_0 \models \neg D\theta$. The detailed proof is given in the Appendix.

As a main result, we obtain completeness and independence from the selection rule (of the operational semantics of Definition 4.2 w.r.t. the declarative semantics of Definition 3.2).

Theorem 4.14 (Completeness) *Assume that the Herbrand universe is finite. Consider a safe program $(P, \mathcal{T})$, a safe goal $G = C_0, \overline{L}$ (where C_0 is the constraint of G), and a selection rule $\mathcal{R}$. Let θ be a grounding substitution for the variables of G such that $C_0\theta$ is satisfiable.*

1. *If $(P, \mathcal{T}) \models_{\mathrm{wf}} \overline{L}\theta$ then there exists a t-tree (of a finite rank) for G via $\mathcal{R}$ with an answer C such that $\mathcal{T} \models C\theta$.*

[14] By Lemma 4.10, $\neg C \vee \neg D$ is a negative answer (the negation of an answer) for C, A. Hence according to Definition 4.2, the constraint of the child of G' is $C, (\neg C \vee \neg D)$, which is equivalent to $C, \neg D$.

[15] For P and G. If G consists of a single atom A then the constraints are in solved form for P and any goal.

2. *If* $(P, \mathcal{T}) \models_{\text{wf}} \neg\overline{L}\theta$ *then there exists a tu-tree (of a finite rank) for* G *via* $\mathcal{R}$ *with a negative answer* C *such that* $\mathcal{T} \models C\theta$.

Proof As the Herbrand universe is finite, the set of possible programs P/M_0 is finite. Thus there exists a natural number k such that $\Psi^k_{P/M_0}(\emptyset)$ is the well-founded model of P/M_0 for each M_0. Now the Theorem follows from Lemma 4.13. □

5 Decidability

Now we show that the well-founded semantics for hybrid programs is decidable in the case of Datalog, i.e. when the set $\mathcal{F}$ of function symbols is a finite set of constants (no safeness condition is needed). The proof employs the soundness and completeness results from the previous sections.

Theorem 5.1 (Decidability) *Assume that the Herbrand universe is finite, and that for any closed constraint* C *it is decidable whether* $\mathcal{T} \models C$. *There exists an algorithm which, for a hybrid program* $(P, \mathcal{T})$ *and a ground atom* A, *finds out whether* $(P, \mathcal{T}) \models_{\text{wf}} A$ *and whether* $(P, \mathcal{T}) \models_{\text{wf}} \neg A$.

Proof We first show that each maximal t- and tu-tree for program $(ground(P), \mathcal{T})$ can be represented and constructed in a finite way.

We say that a constraint is in *ground solved form* if it is built out of the constraints of the rules of $ground(P)$ by means of $\neg$ and $\wedge$. The set of such constraints up to logical equivalence is finite (convert them to a disjunctive normal form, remove repeated literals in the conjunctions, remove repeated conjunctions).

Let B be a ground rule atom, and Tr be a t-tree or a tu-tree for B. Each satisfiable equality constraint that appears in the tree is of the form $a = a$ (where $a \in \mathcal{F}$), hence it is valid and may be removed. Assume that each lower rank (negative) answer employed in the tree is in ground solved form. Then each constraint in the tree is in ground solved form. So the set of these constraints is finite up to equivalence.

The set of rule literals that appear in Tr is a subset of the literals of $ground(P)$. Thus the set is finite. So the set of conjunctions of such literals is finite, up to logical equivalence (repeated occurrences of a literal can be removed). As a result, we obtain that the set of nodes of Tr is finite up to logical equivalence.

Assume now that the (negative) answers from lower rank trees employed in Tr are known. Then a finite *representation* of Tr can be constructed top-down starting from the root B. Before adding a node G to the current (sub-graph of the) tree, it is checked whether a node G' logically equivalent to G already exists. If it does and the same literal is selected in both nodes then G is not added. This process terminates, its result is a tree Tr' which is a finite sub-graph of Tr. Each successful leaf of Tr is (logically equivalent to) a successful leaf of Tr'. In the terminology of the previous section, the disjunction of the successful leaves of Tr' is a finite (pseudo-) answer of Tr (and is logically equivalent to each finite (pseudo-) answer of Tr).

In this way, finite representations of the maximal t- and tu-trees for all the atoms can be constructed, first for rank 0 and then stepwise for ranks $1, 2, \ldots$. The process is terminated at rank k when for each atom A the obtained finite (pseudo-) answers of rank $k-1$ and k are equivalent (as it is then a finite (pseudo-) answer for A of any rank $> k$).

The well-founded models of P and those of $ground(P)$ are the same; hence the programs are equivalent: $(P, \mathcal{T}) \models_{\text{wf}} F$ iff $(ground(P), \mathcal{T}) \models_{\text{wf}} F$ for any formula F. As $ground(P)$ is finite and safe, the completeness Lemma 4.13 applies. Thus, for any selection rule $\mathcal{R}$,

- if $(P, \mathcal{T}) \models_{\mathrm{wf}} A$ then $\mathcal{T} \models C$, for some finite $k \geq 0$ and any maximal t-tree of rank $\geq k$ for A and $(ground(P), \mathcal{T})$, with a finite answer C,
- if $(P, \mathcal{T}) \models_{\mathrm{wf}} \neg A$ then $\mathcal{T} \models \neg D$, for some finite $k \geq 0$ and any maximal tu-tree of rank $\geq k$ for A and $(ground(P), \mathcal{T})$, with a finite pseudo-answer D.

Hence, by the soundness Theorem 4.9, $(P, \mathcal{T}) \models_{\mathrm{wf}} A$ iff $\mathcal{T} \models C$ (respectively $(P, \mathcal{T}) \models_{\mathrm{wf}} \neg A$ iff $\mathcal{T} \models \neg D$) for the finite answer C (pseudo-answer D) for A computed earlier. Checking whether $\mathcal{T} \models \neg C$ and $\mathcal{T} \models \neg D$ is decidable by the assumptions of the theorem. □

The decidability result should be compared with the fact that under the assumption of Thoerem 5.1 it is undecidable whether $P \cup \mathcal{T} \models A$ [26]. The difference is that the logical consequence deals with arbitrary interpretation domains, while the semantics of hybrid programs interprets P over the Herbrand universe, which in this case is finite.

6 Related work

The notions of external theory and constraints used in hybrid rules are similar to those used in CLP. However, classical CLP does not support non-monotonic reasoning. The results presented in this paper can thus also be seen as an approach to integration of both paradigms, based on the well-founded semantics of normal programs.

We are aware of few papers on non-monotonic negation for CLP. The approach of [20] employs the 3-valued completion semantics. It is similar to the work presented here; in both cases the operational semantics follows the idea of Ref. [12] of selecting a cross-section of a derivation tree and negating the disjunction of respective constraints. The completion semantics is also used in Ref. [39]. An approach generalizing the well-founded semantics for CLP is presented in Ref. [9]. It, however, assigns the semantics only to some programs (those that can be transformed to an irreducible program), while our semantics deals with all programs. In contrast to our approach, the operational semantics of Ref. [9] is not top-down and goal driven. It consists of applying program transformations to obtain an irreducible program; the latter can be used to obtain answers to goals.

The presented framework makes it possible to integrate normal logic programs with ontologies expressed as first-order theories, including those specified in the web ontology languages OWL-DL and OWL-Lite.

The problem of integration of rules and ontologies has been addressed in different ways. One line of research aims at achieving the integration by embedding rules, ontologies and their combinations in a known logic. A well-known proposal of this kind is SWRL [24], extending ontologies with Horn formulae within FOL, but not allowing non-monotonic rules. More recent attempts [7,8,31] address the issue of non-monotonicity by embedding non-monotonic rules and DL ontologies in various logics which make it possible to capture non-monotonicity.

In contrast to that, we achieve the integration by a direct definition of the semantics of hybrid rules. The declarative semantics combines the FOL semantics of the external theories with the well-founded semantics of logic programs. Negation in the constraints of the external theory is interpreted in the classical way, while negation in rule literals is non-monotonic. We now compare our work with the approaches to integration of rules and ontologies which make similar assumptions. We note first that all related work of this kind is based on Datalog rules, while our approach admits non-nullary function symbols.

Our work is strongly motivated by the early $\mathcal{AL}$-log approach [10] where positive Datalog was extended by allowing the concepts of $\mathcal{ALC}$ DL as constraints in safe Datalog rules. The operational semantics of $\mathcal{AL}$-log relies on an extension of SLD-resolution where the disjunction of constraints from different derivations is to be submitted for validity check to the DL-reasoner. We adopted the $\mathcal{AL}$-log idea of extending rules with constraints in the body, and applied it to more expressive rules including non-monotonic negation, and to arbitrary external theories of FOL.

In our approach, the heads of the hybrid rules are atoms built with rule predicates. Thus the semantics of the rule predicates depends on the external theory which is assumed to be given a priori and not to depend on the rules. The rationale for that is that the rules describe a specific application while the theory (for example an ontology) provides a knowledge common for an application domain. In contrast to that, several papers [32,35,36] allow the use of ontology predicates in the heads of rules, defining thus an integrated language where rule predicates and ontology predicates may be mutually dependent, and ontology predicates can be (re-) defined by rules.

The paper [32] defines *DL rules*, a decidable combination of OWL-DL with disjunctive Datalog without non-monotonic negation. In contrast to that, our primary concern is non-monotonic reasoning.

The r-hybrid knowledge bases [35] and $\mathcal{DL}+log$ [36] are based on disjunctive Datalog with non-monotonic negation under the stable model semantics. The objective is to define a generic integration scheme of this variant of Datalog with an arbitrary Description Logic. The DL-rules defined under this scheme may include DL predicates not only in their bodies but also in the heads. A hybrid $\mathcal{DL}+log$ knowledge base consists of a DL knowledge base $\mathcal{K}$ and a set of hybrid rules $\mathcal{P}$. A notion of *non-monotonic model* of such a knowledge base is defined by referring to first-order models[16] of $\mathcal{K}$ and to the stable models of disjunctive Datalog. Similarly to our declarative semantics, models of $\mathcal{K}$ are used to transform the set of grounded hybrid rules into a set of ground Datalog rules, not including DL-atoms. However, as the heads of the hybrid rules may include DL-atoms, the transformation is more elaborate than our $\mathcal{P}/M_0$ transformation. Also the semantics of $\mathcal{DL}+log$ is based on stable models of the transformed ground rules, while our semantics is based on the well-founded semantics of $\mathcal{P}/M_0$. For stratified normal logic programs the stable model semantics is equivalent to the well-founded semantics [3]. Thus for stratified sets of rules $\mathcal{DL}+log$ coincides with our approach (provided the rules are non- disjunctive, and without DL-atoms in their heads, and the external theory satisfies the requirements of $\mathcal{DL}+log$).

The proposed reasoning algorithm (NMSAT-$\mathcal{DL}+log$) works bottom-up and is based on grounding. Our operational semantics works top-down and does not require grounding. Decidability of $\mathcal{DL}+log$ is achieved by a weak safeness condition. The condition is similar to that in our approach. However, we do not need it for decidability (but for completeness of the operational semantics; it is also one of alternative sufficient conditions for soundness).

The language of Description Logic Programs (dl-programs) [18] integrates OWL DL with Datalog rules with negation. This is done by allowing in the bodies so called *dl-queries* to a given ontology. The queries may locally modify the ontology. Two kinds of declarative semantics are considered for the integrated language. The semantics of choice extends the stable model semantics of Datalog with negation[17] [18] but an extension of the well-founded

[16] However, only a fixed domain of interpretation is considered, with a fixed interpretation of constants. Moreover, the interpretation is a bijection (each domain element is denoted by a distinct constant), and the domain is countably infinite.

[17] More recent versions of this work are based on disjunctive Datalog with negation. A further extension is *HEX-programs* [17], where *external atoms* are used to model interface with arbitrary external computations.

semantics is also considered [19]. In both variants of the declarative semantics the *truth value* of a rule w.r.t. to an interpretation depends on dl-queries in the rule being *logical consequences* of the respective ontologies. This makes the semantics incompatible with the standard semantics of the first-order logic. For example consider rules $P = \{ p \leftarrow Q_1;\ p \leftarrow Q_2 \}$, where none of DL-formulae Q_1, Q_2 is a logical consequence of the ontology $\mathcal{T}$, but in each model of $\mathcal{T}$ at least one of them is true. Then p is a logical consequence of $P \cup \mathcal{T}$, but will not follow from $(P, \mathcal{T})$ represented as a dl-program. In contrast to that, our approach is compatible with FOL, in the sense explained in Sect. 3.2. In the last example, p follows from the hybrid program $(P, \mathcal{T})$.

7 Conclusions

We presented a framework for integration of normal logic programs under the well-founded semantics with first-order theories. Syntactically the integration is achieved by extending the bodies of normal clauses with (certain) formulae of a given external theory, resulting in the notions of hybrid rule and hybrid program. It is assumed that the theories are external sources of knowledge and are not modified during the integration. Therefore, it is required that the predicates of the extended normal program and the predicates of the external theory are distinct.

The main contributions of this work are as follows:

- The declarative semantics of hybrid programs, combining the (3-valued) well-founded semantics of normal programs with the (2-valued) logical semantics of the external theory. It combines non-monotonic negation of the well-founded semantics with the classical negation of FOL. It allows non-constant function symbols. In contrast to most of related approaches it is defined for arbitrary external FOL theories.
  The declarative semantics is undecidable, however, it is decidable for Datalog hybrid programs with decidable external theories. In the special case of a hybrid program $(P, \mathcal{T})$ where $\mathcal{P}$ does not include negation, the declarative semantics is compatible with the semantics of FOL, in the sense explained in Sect. 3.2. The semantics makes possible reasoning by cases (cf. Example 2.3).
- The operational semantics that describes how to answer (not necessarily ground) conjunctive queries. This includes handling of non-ground negative queries by combination of the ideas of CLP [30] and constructive negation of Ref. [12]. The operational semantics can be seen as an extension of SLS-resolution with handling of non-ground negative queries and constraints of the external theory. It can be implemented by compilation to Prolog [14,15], and employing an LP reasoner for the well-founded semantics and a reasoner for the external theory. So the implementation requires rather small efforts thanks to re-using the existing reasoners. A prototype implementation of this kind is described in Ref. [14]. It allows non- constant function symbols, the external theories are OWL ontologies. It uses XSB Prolog [37] as an LP engine, and can use various OWL reasoners. It is efficient, in the sense that the number of queries to the OWL reasoner is small.
- Soundness and completeness results: the operational semantics was shown to be sound w.r.t. the declarative semantics. It is complete (and independent from the selection rule) for safe hybrid programs where the function symbols are constants.

The external theory can be a theory of a constraint domain of CLP. Thus our framework additionally provides a method of adding non-monotonic negation to CLP, in a way generalizing the well-founded semantics.

In contrast to most of related work, our approach works for arbitrary external FOL theories. There are no restrictions on the alphabet of function symbols (it may be finite or infinite). We do not impose any conditions on the equality in the external theories, like unique name assumption (UNA) or freeness axioms (CET). There are certain requirements related to the operational semantics; however, we show how an arbitrary external theory $\mathcal{T}'$ can be extended to a theory $\mathcal{T}$ satisfying the requirements.

Acknowledgments This research has been partially funded by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779(cf. http://rewerse.net).

Appendix

Here we present proofs of Lemma 4.7 and of two main technical lemmas from Sects. 4.3 and 4.4. By $var(F)$ we denote the set of free variables of a formula F.

Proof of Lemma 4.7 We first prove part 2 by induction. Then part 1 follows immediately. Assume that a node G_i is safe apart of V. We show that each its child is safe apart of V.

Let $G_i = C, \overline{L}, p(\overline{t}), \overline{L'}$ and its child $G_{i+1} = \overline{t} = \overline{u}, C, C', \overline{L}, \overline{K}, \overline{L'}$ be derived from G_i by a rule $R = p(\overline{u}) \leftarrow C', \overline{K}$. Let $\overline{N}$ be the negative literals of $\overline{L}, \overline{L'}$, and $\overline{N'}$ be the negative literals of $\overline{K}$. Each free variable of $\overline{u}, C', \overline{N'}$, is bound in C' to a ground term or to a variable occurring in a positive literal of $\overline{K}$ (as R is safe). Each variable from $var(C, \overline{N})\backslash V$ is bound in C to a ground term or to a variable occurring in a positive literal of $\overline{L}, \overline{L'}$ or in $\overline{t}$ (as G_i is safe apart of V). Hence each variable from $var(C, \overline{N}, \overline{t} = \overline{u}, C', \overline{N'})\backslash V$ is bound in $C, \overline{t} = \overline{u}, C'$ to a ground term or to a variable occurring in a positive literal of $\overline{K}, \overline{L}, \overline{L'}$.

Let $G_i = C, \overline{L}, \neg A, \overline{L'}$ with a negative literal $\neg A$ selected, with a child $G_{i+1} = C, C', \overline{L}, \overline{L'}$. We have $var(C') \subseteq var(C, A)$, thus $var(C, C')\backslash V \subseteq var(C, A)\backslash V$. Thus each variable from $var(C, C')\backslash V$ is bound in C, C' to a ground term or to a variable occurring in a positive literal of $\overline{L}, \overline{L'}$. As the negative literals of G_{i+1} are those of G_i, we obtain that G_{i+1} is safe apart of V. □

The proofs below refer to ground programs of the form P/M_0 (cf. Definition 3.2), and the operator defining the well-founded semantics of normal logic programs (cf. Sect. 2.1):

$$\Psi_{P/M_0}(I) = (\mathcal{M}_{(P/M_0)/_t I} \cap \mathcal{H}) \cup \neg(\mathcal{H}\backslash\mathcal{M}_{(P/M_0)/_{tu} I})$$

Remember that from monotonicity of Ψ_{P/M_0} it follows that $k' \leq k$ implies $\Psi^{k'}_{P/M_0}(\emptyset) \subseteq \Psi^{k}_{P/M_0}(\emptyset)$.

Proof of Soundness Lemma 4.8 By transfinite induction on the rank of the t-tree and tu-tree. Assume that the lemma holds for t-trees and tu-trees of rank $< k$. We can assume that θ binds only the free variables of G.

1. Assume that C is an answer of a t-tree for G of rank k, and that $M_0 \models C\theta$. Consider the branch $G_1, \ldots, G_m$ of the tree from the root G to the leaf C' (i.e. $G_1 = G$, $G_m = C'$) such that $C = (\ldots \vee C' \vee \ldots)|_G$ and $M_0 \models (C'|_G)\theta$. We show that some ground instance of $C'\theta$ is true in M_0. This is obvious when M_0 is a Herbrand interpretation or when the

set of constraints has the witness property. If P is safe then C' is safe apart from the set $V_0 = var(G)$, by Lemma 4.7. So each variable $x \in var(C')\backslash V_0$ is bound in C' to a ground term t_x. Consider a substitution $\varphi = \{ x/t_x \mid x \in var(C')\backslash V_0 \}$. Formula $C'|_G$ is equivalent to $(C'\varphi)|_G$ and to $C'\varphi$. Hence $M_0 \models C'\varphi\theta$. The latter formula is a ground instance of $C'\theta$, as $\varphi\theta = \theta\varphi$.

Thus substitution θ can be extended to a substitution θ' such that $M_0 \models C'\theta'$, and $G\theta' = G\theta$. Notice that θ' is applicable to all the goals in the branch (as for any such goal its constraint is of the form $C_1, \ldots, C_l$, where $C' = C_1, \ldots, C_n,\ l \leq n$).

The sequence $G_1\theta', \ldots, G_m\theta'$ is a successful branch of an instance of the considered t-tree for G. If a positive literal is selected in G_i then $G_{i+1}\theta'$ is derived from $G_i\theta$ by a rule from (P/M_0). Whenever a negative literal $\neg A$ is selected in G_i and C_i is the constraint of G_i then the constraint of G_{i+1} is C_i, C'', where C'' is a negative answer of a tu-tree of rank $j < k$ for C_i, A. As θ' is applicable to G_{i+1}, we have $M_0 \models (C_i, C'')\theta'$. By the inductive assumption, $\Psi^{j+1}_{P/M_0}(\emptyset) \models_3 \neg A\theta'$. The same holds for any $k' \geq j$.

Thus the sequence $G_1\theta', \ldots, G_m\theta'$ is a successful SLD-derivation for $G\theta'$ and the program $P_t = (P/M_0)/_t \Psi^{k'+1}_{P/M_0}(\emptyset)$ for some $k' < k$ (if $k = 0$ then $P_t = P/M_0$). Hence $\mathcal{M}_{P_t} \models G\theta$, by soundness of SLD-resolution [2]. By the definition of Ψ_{P/M_0}, for each positive literal A' of $G\theta$ we have $\Psi^{k'+2}_{P/M_0}(\emptyset) \models_3 A'$. For each negative literal $\neg A'$ of $G\theta$ we have $\neg A' \in P_t$, i.e. $\Psi^{k'+1}_{P/M_0}(\emptyset) \models_3 \neg A'$. By monotonicity of Ψ_{P/M_0} for each literal L of $G\theta$ we have $\Psi^{k+1}_{P/M_0}(\emptyset) \models_3 L$. Hence $\Psi^{k+1}_{P/M_0}(\emptyset) \models_3 G\theta$.

2. Consider a tu-tree Tr of rank k for G. Let the negative answer C be obtained from Tr and its cross-section F. Let CF be the constraints of the nodes of F. So C implies $\neg(C_i|_G)$ for each $C_i \in CF$. Assume that $M_0 \models C\theta$. Then $M_0 \not\models C_i\theta'$, for any extension θ' of θ grounding the variables of C_i and for each $C_i \in CF$. Thus, in a ground instance $Tr\theta$ of the tree, no node corresponds to a node of F (nor to a descendant of a node of F). Hence no leaf of $Tr\theta$ is successful.

   Out of $Tr\theta$ we construct an SLD-tree with root $G\theta$ for a certain ground program. The tree has no success leaves, hence $G\theta$ is false in the least Herbrand model of the program (by completeness of SLD-resolution).

   In $Tr\theta$, consider a leaf $G'\theta'$ corresponding to a node $G' = (C', \overline{L}, \neg A, \overline{L'})$ of Tr, with a negative literal $\neg A$ selected. The node G' does not have a child $C', \overline{L}, \overline{L'}$ (such a child implies that $G'\theta'$ is not a leaf). Hence case 4(b)ii or 4(b)iii has been applied to the node. Thus there exists an answer C'' of a t-tree for C', A of rank $k' < k$. Moreover, $(\neg C'', C')\theta'$ is false in M_0 (otherwise $\neg C'', C'$ is satisfiable, G' has a child $\neg C'', C', \overline{L}, \overline{L'}$, and $G'\theta'$ has a child $\overline{L}\theta', \overline{L'}\theta'$). Hence $\neg C''\theta'$ is false in M_0 (as $C'\theta'$ is true in M_0). From $M_0 \models C''\theta'$ and from the inductive assumption we obtain $\Psi^{k'+1}_{P/M_0}(\emptyset) \models_3 A\theta'$. By monotonicity of Ψ_{P/M_0} the same holds for any $k'' \geq k'$. Thus $\neg A\theta' \notin (P/M_0)/_{tu} \Psi^{k'+1}_{P/M_0}(\emptyset)$, by the definition of $/_{tu}$. As $k' + 1 \leq k$, we have $\neg A\theta' \notin (P/M_0)/_{tu} \Psi^{k}_{P/M_0}(\emptyset)$. Hence the node $G'\theta'$ (with $\neg A\theta'$ selected) is a leaf of an SLD-tree for program $(P/M_0)/_{tu} \Psi^{k}_{P/M_0}(\emptyset)$. Notice that this reasoning is also valid for a non-finite k. In particular, $k' + 1 < k$ if k is a limit ordinal.

   If a non-leaf node $G'\theta'$ in $Tr\theta$ corresponds to $G' = (C', \overline{L}, \neg A_{G'}, \overline{L'})$ of Tr, with $\neg A_{G'}$ selected, then the child of $G'\theta'$ is $(\overline{L}, \overline{L'})\theta'$. For each such node $G'\theta'$ of $Tr\theta$, let us add $\neg A_{G'}\theta'$ to the child and all its descendants. The obtained tree is an SLD-tree for $G\theta$ and a program $P_{tu} \cup P_{\text{void}}$, where $P_{tu} = (P/M_0)/_{tu} \Psi^{k}_{P/M_0}(\emptyset)$, and each clause of P_{void} is of the form $\neg A \leftarrow \neg A$. Adding P_{void} to a definite clause logic program does not change its

least Herbrand model. The obtained tree does not have a success node. By completeness of SLD-resolution [2] $\mathcal{M}_{P_{tu}} \not\models G\theta$, and for some literal L of $G\theta$ we have $L \notin \mathcal{M}_{P_{tu}}$. If L is of the form $\neg A$ then $A \in \Psi^k_{P/M_0}(\emptyset)$ (by the definition of $/_{tu}$), and L is $\mathbf{f}$ in $\Psi^k_{P/M_0}(\emptyset)$. Otherwise $L \in \mathcal{H}\backslash\mathcal{M}_{P_{tu}}$ and $\neg L \in \Psi^{k+1}_{P/M_0}(\emptyset)$, hence L is $\mathbf{f}$ in $\Psi^{k+1}_{P/M_0}(\emptyset)$. In both cases L is $\mathbf{f}$ in $\Psi^{k+1}_{P/M_0}(\emptyset)$, thus $\Psi^{k+1}_{P/M_0}(\emptyset) \models_3 \neg G\theta$. □

Proof of Completeness Lemma 4.13 If $\Psi^k_{P/M_0}(\emptyset) \models_3 G\theta$ or $\Psi^k_{P/M_0}(\emptyset) \models_3 \neg G\theta$ then we say that $k-1$ is a level of $G\theta$ when G does not contain a negative literal and k is a level of $G\theta$ when G contains a negative literal. The proof is by induction on a level of $G\theta$; the rank of the constructed t- or tu-tree for G is the level of $G\theta$.

Let $G\theta$ be of level $k' \in \{k-1,\ k\}$. Let us denote $I = \Psi^{k-1}_{P/M_0}(\emptyset)$ and $J = \Psi^{k'}_{P/M_0}(\emptyset)$. So $J = \Psi_{P/M_0}(I)$ if G contains a negative literal, and $J = I$ otherwise. In both cases $I \subseteq J$.

1. Assume $\Psi^k_{P/M_0}(\emptyset) \models_3 G\theta$. Each positive literal A in $G\theta$ is a member of $\mathcal{M}_{(P/M_0)/_t I}$ (by the definition of Ψ) and hence of $\mathcal{M}_{(P/M_0)/_t J}$. For each negative literal $\neg A$ in $G\theta$, we have $J \models_3 \neg A$ (as then $\Psi^k_{P/M_0}(\emptyset) = J$). Thus $\neg A \in (P/M_0)/_t J$, by the definition of $/_t$. So, by completeness of SLD-resolution, for any selection rule there exists a successful SLD-derivation $\mathcal{D}$ for the goal $G\theta$ and program $(P/M_0)/_t J$.
   We now show how negative literals in the derivation are related to lower rank tu-trees. Consider a negative literal $\neg A$ occurring in $\mathcal{D}$. We have $\neg A \in (P/M_0)/_t J$ (i.e. $\neg A \in J$), so A is of level $k'-1$. By the inductive assumption, for any goal B such that $A = B\theta'$ is a ground instance of B, if C' is the negation of a finite pseudo-answer of a maximal tu-tree for B via $\mathcal{R}$ of rank $k'-1$ then $M_0 \models C'\theta'$.
   Consider a maximal t-tree Tr of rank k' for G and $(P, \mathcal{T})$ via $\mathcal{R}$. Let $Tr\theta$ be a ground instance of Tr. It can be seen as an SLD-tree via a selection rule $\mathcal{R}'$, for a certain ground definite program containing the rules of P/M_0 and some facts of the form $\neg A$. For $\mathcal{R}'$ there exists a derivation $\mathcal{D}$ as above. We show that $\mathcal{D}$ is a branch of $Tr\theta$. We prove by induction on i that the i-th goal of $\mathcal{D}$ is a node of $Tr\theta$. Let a goal $G'\theta'$ of $\mathcal{D}$ be a node of $Tr\theta$, corresponding to a node G' of Tr. If a positive literal is selected in $G'\theta'$ and in G' then any goal derived from $G'\theta'$ by a rule from P/M_0 is a ground instance $G''\theta''$ of a child G'' of G' in Tr, where θ'' is an extension of θ'. If a negative literal $\neg A$ is selected in $G'\theta'$ then a $\neg B$ is selected in G' and $\neg A = \neg B\theta'$. The tree Tr refers to a maximal tu-tree for B, and to the negation C' of its finite pseudo-answer. As explained in the previous paragraph, $M_0 \models C'\theta'$ follows from the inductive assumption. The child G'' of G' in Tr is G' with $\neg B_i$ removed and C' added. So substitution θ' is applicable to G'', and $G''\theta'$ is $G'\theta'$ with $\neg A_i$ removed. Thus the successor of $G'\theta'$ in $\mathcal{D}$ is the child $G''\theta'$ of $G'\theta'$ in $Tr\theta$.
   The last goal of $\mathcal{D}$ is empty, it is a ground instance of a leaf C_s of Tr. Thus $M_0 \models (C_s\theta)|_G$. Let C be a finite answer of the t-tree. $M_0 \models C\theta$.
2. Assume that $\Psi^k_{P/M_0}(\emptyset) \models_3 \neg G\theta$. Some literal L of $G\theta$ is false in $\Psi^k_{P/M_0}(\emptyset) = \Psi_{P/M_0}(I)$. If $L = A$ is an atom then $\neg A \in \Psi_{P/M_0}(I) \subseteq \Psi_{P/M_0}(J)$. Hence $A \notin \mathcal{M}_{(P/M_0)/_{tu} J}$ (by the definition of Ψ_{P/M_0}). If $L = \neg A$ is a negative literal then $A \in \Psi_{P/M_0}(I) = J$. Hence $\neg A \notin (P/M_0)/_{tu} J$ (by the definition of $/_{tu}$). In both cases, $L \notin \mathcal{M}_{(P/M_0)/_{tu} J}$. Thus, by soundness of SLD-resolution, there does not exist a successful SLD-derivation for $G\theta$ and the program $(P/M_0)/_{tu} J$ (as otherwise $L \in \mathcal{M}_{(P/M_0)/_{tu} J}$, contradiction).
   Consider a maximal tu-tree Tr of rank k' for G via $\mathcal{R}$, with a pseudo-answer D. We have to show that $M_0 \models \neg D\theta$. Assume the contrary; then Tr has a success leaf C_s such that $M_0 \models (C_s|_G)\theta$. Consider a branch of Tr from G to C_s. Let $G_1, \ldots, G_l$ be the goals of

the branch ($G = G_1, G_l = C_s$). As P is safe, by Lemma 4.7 there exists a substitution θ' which is an extension of θ and is applicable to the nodes of the branch (see part 1 of the proof of Lemma 4.8 for details). We show that $G_1\theta', \ldots, G_l\theta'$ is a successful SLD-derivation for $G\theta$ and $(P/M_0)/_{tu}J$, which is a contradiction.

If a positive literal is selected in G_i and a rule $r \in P$ is applied to G_i to obtain G_{i+1} then $G_{i+1}\theta'$ is obtained from $G_i\theta'$ by applying a normal ground instance $r\sigma \in (P/M_0)/_{tu}J$ of r.

Assume a negative literal $\neg B$ is selected in G_i. The constraint of G_{i+1} is equivalent to $C_i, \neg C''$ where C_i is the constraint of G_i and C'' is a finite answer of a maximal t-tree of rank $k' - 1$ for B. We have $M_0 \models \neg C''\theta'$, as θ' is applicable to G_{i+1}. Suppose $B\theta' \in J$. This leads to contradiction: such $B\theta'$ is of level $k' - 1$ and, by the inductive assumption, the answer C'' satisfies $M_0 \models C''\theta'$. Thus $B\theta' \notin J$ and $\neg B\theta' \in (P/M_0)/_{tu}J$. Thus $G_{i+1}\theta'$ is obtained from $G_i\theta'$ by applying a rule from $(P/M_0)/_{tu}J$.

Hence $G_1\theta', \ldots, G_l\theta'$ is a successful SLD-derivation for $G\theta$ and $(P/M_0)/_{tu}J$; the assumption that $M_0 \models (C_s|_G)\theta$ is false. So $M_0 \models \neg(C_s|_G)\theta$ for each success leaf C_s of Tr. Let D be a finite pseudo-answer of Tr. Then $M_0 \models \neg D\theta$ for a negative answer $\neg D$ of the tree. □

References

1. Antoniou G, Damásio CV, Grosof B, Horrocks I, Kifer M, Małuszyński J, Patel-Schneider PF (2005) Combining rules and ontologies. a survey. FP6 NoE REWERSE, Deliverable I3-D3, Available at. http://rewerse.net/deliverables/m12/i3-d3.pdf
2. Apt KR (1997) From logic programming to prolog, International series in computer science. Prentice-Hall, Englewood Cliffs
3. Apt KR, Bol RN (1994) Logic programming and negation: a survey. J Log Program 19/20:9–71
4. Baader F, Calvanese D, McGuiness D, Nardi D, Patel-Schneider P (eds) (2003) The description logic handbook. Cambridge University Press, Cambridge
5. Baral C, Gelfond M (1994) Logic programming and knowledge representation. J Log Program 19/20:73–148
6. Clark KL (1978) Negation as failure. In: Gallaire H, Minker J (eds) Logic and databases. Plenum Press, New York, pp 293–322
7. de Bruijn J, Eiter T, Tompits H (2008) Embedding approaches to combining rules and ontologies into autoepistemic logic. In: Brewka G, Lang J (eds) KR. AAAI Press, pp 485–495
8. de Bruijn J, Pearce D, Polleres A, Valverde A (2007) Quantified equilibrium logic and hybrid rules. In: Marchiori et al. [29] pp 58–72
9. Dix J, Stolzenburg F (1998) A framework to incorporate non-monotonic reasoning into constraint logic programming. J Log Program 37(1–3):47–76
10. Donini F, Lenzerini M, Nardi D, Schaerf A (1998) AL-Log: integrating datalog and description logics. Int Inf Syst 10(3):227–252
11. Drabent W (1993) SLS-resolution without floundering. In: Pereira LM, Nerode A (eds) Proceedings of 2nd international workshop on logic programming and non-monotonic reasoning. MIT Press, pp 82–98
12. Drabent W (1995) What is failure? An approach to constructive negation. Acta Inform 32(1):27–59
13. Drabent W, Eiter T, Ianni G, Krennwallner T, Lukasiewicz T, Małuszyński J (2009) Hybrid reasoning with rules and ontologies. In: Bry F, Małuszyński J (eds) Semantic techniques for the web. The REWERSE perspective, vol 5500 of lecture notes in computer science. Springer, pp 1–49
14. Drabent W, Henriksson J, Małuszyński J (2007a) HD-rules: a hybrid system interfacing prolog with DL-reasoners. In: Proceedings of the ICLP'07 workshop on applications of logic programming to the web, semantic web and semantic web services (ALPSWS2007), CEUR workshop proceedings, vol 287. http://www.ceur-ws.org/Vol-287. Updated version of (Drabent et al. [15])
15. Drabent W, Henriksson J, Małuszyński J (2007b) Hybrid reasoning with rules and constraints under well-founded semantics, In: Marchiori et al. [29] pp 348–357
16. Drabent W, Małuszyński J (2007) Well-founded semantics for hybrid rules. In: Marchiori et al. [29] pp 1–15

17. Eiter T, Ianni G, Schindlauer R, Tompits H (2006) Effective integration of declarative rules with external evaluations for semantic-web reasoning. In: Sure Y, Domingue J (eds) ESWC vol 4011 of lecture notes in computer science, Springer, pp 273–287
18. Eiter T, Lukasiewicz T, Schindlauer R, Tompits H (2004a) Combining answer set programming with description logics for the semantic web. In: Proceedings of the international conference of knowledge representation and reasoning (KR'04). http://citeseer.ist.psu.edu/eiter04combining.html
19. Eiter T, Lukasiewicz T, Schindlauer R, Tompits H (2004b) Well-founded semantics for description logic programs in the semantic web. In: Antoniou G, Boley H (eds) RuleML vol 3323 of lecture notes in computer science. Springer, pp 81–97
20. Fages F (1997) Constructive negation by pruning. J Log Program 32(2):85–118
21. Ferrand G, Deransart P (1993) Proof method of partial correctness and weak completeness for normal logic programs. J Log Program 17(2/3&4):265–278
22. Gelfond M, Lifschitz V (1988) The stable model semantics for logic programming. In: Kowalski RA, Bowen K (eds) Proceedings of the fifth international conference on logic programming. The MIT Press, Cambridge, Massachusetts, pp 1070–1080
23. Gruber TR (1995) Towards principles for the design of ontologies used for knowledge sharing. J Hum Comput Stud 43:907–928
24. Horrocks I, Patel-Schneider PF (2004) A proposal for an OWL rules language. WWW '04: Proceedings of the 13th international conference on World Wide Web. ACM, New York, NY, USA, pp 723–731
25. Kunen K (1987) Negation in logic programming. J Log Program 4(4):289–308
26. Levy A, Rousset M-C (1998) Combining horn rules and description logics in carin. Artif Intell 104(1–2): 165–209
27. Lloyd JW (1987) Foundations of logic programming, second extended edn. Springer, Berlin
28. Małuszyński J (2009) Integration of rules and ontologies. In: Liu L, Özsu MT (eds) Encyclopedia of database systems. Springer US, pp 1546–1551
29. Marchiori M, Pan JZ, de Sainte Marie C (eds) (2007) Web reasoning and rule systems, first international conference, RR 2007, Innsbruck, Austria, June 7–8, 2007, Proceedings, vol 4524 of lecture notes in computer science, Springer
30. Marriott K, Stuckey PJ, Wallace M (2006) Constraint logic programming. Handbook of constraint programming. Elsevier, Amsterdam
31. Motik B, Rosati R (2007) A faithful integration of description logics with logic programming. In: Veloso MM (ed) IJCAI 2007, Proceedings of the 20th international joint conference on artificial intelligence. pp 477–482
32. Motik B, Sattler U, Studer R (2005) Query answering for OWL-DL with rules. J Web Sem 3(1):41–60
33. Patel-Schneider PF, Hayes P, Horrocks I (2004) OWL web ontology language semantics and abstract syntax, W3C Recommendation. Available at. http://www.w3.org/TR/owl-semantics/
34. Przymusinski TC (1989) On the declarative and procedural semantics of logic programs. J Autom Reason 5:167–205
35. Rosati R (2005) On the decidability and complexity of integrating ontologies and rules. J Web Semant 3:61–73
36. Rosati R (2006) $\mathcal{DL}+log$: tight integration of description logics and disjunctive datalog. In: Doherty P, Mylopoulos J, Welty CA (eds) KR. AAAI Press, Menlo Park, pp 68–78
37. Sagonas C, Swift T, Warren DS et al. (2007) The XSB system. Volume1: programmer's manual. Available at. http://xsb.sourceforge.net
38. Shoenfield JR (1967) Mathematical logic. Addison-Wesley, Reading
39. Stuckey PJ (1995) Negation and constraint logic programming. Inf Comput 118(1):12–33
40. van Gelder A, Ross KA, Schlipf JS (1988) Unfounded sets and well-founded semantics for general logic programs. Principles of database systems. ACM, pp 221–230

Author Biographies

Włodzimierz Drabent received a Ph.D. degree from Warsaw University of Technology, and a D.Sc. degree from Institute of Computer Science, Polish Academy of Sciences. He is an associate professor at Institute of Computer Science, Polish Academy of Sciences, and at Department of Computer and Information Science, Linköping University, Sweden. He was a visiting professor at Department of Computer Science, University of California, Riverside. He published numerous technical papers. His research interests are related to logic programming, non-monotonic reasoning, proving program properties, debugging, program analysis, and semantics of programming languages.

Jan Małuszyński is since 2008 Professor Emeritus at the Department of Information and Computer Science of Linköping University, Sweden. His Ph.D. degree is from the Institute of Mathematics of the Polish Academy of Sciences. Until 1982 he held research positions at the Institute of Computer Science, Polish Academy of Science. In 1982 he joined Linköping University where he organized and led the Laboratory for Logic Programming. With main research interests in logic programming and recently also in the semantic web he authored numerous technical papers and edited a number of collections and conference proceedings. He held visiting positions at the Technical University of Denmark, at the University of Utah, and at INRIA Rocquencourt. He served on the editorial boards of Journal of Logic Programming, Theory and Practice of Logic Programming and Journal of Universal Computer Science.