Partial Functions and Equality in Answer Set Programming

Pedro Cabalar[⋆]

Department of Computer Science,
Corunna University (Corunna, Spain),
cabalar@udc.es

EXTENDED VERSION

Abstract. In this paper we propose an extension of Answer Set Programming (ASP) [1], and in particular, of its most general logical counterpart, Quantified Equilibrium Logic (QEL) [2], to deal with partial functions. Although the treatment of equality in QEL can be established in different ways, we first analyse the choice of decidable equality with complete functions and Herbrand models, recently proposed in the literature [3]. We argue that this choice yields some counterintuitive effects from a logic programming and knowledge representation point of view. We then propose a variant called $\text{QEL}^{=}_{\mathcal{F}}$ where the set of functions is partitioned into *partial* and Herbrand functions (we also call *constructors*). In the rest of the paper, we show a direct connection to Scott's *Logic of Existence* [4] and present a practical application, proposing an extension of normal logic programs to deal with partial functions and equality, so that they can be translated into function-free normal programs, being possible in this way to compute their answer sets with any standard ASP solver.

1 Introduction

Since its introduction two decades ago, the paradigm of *Answer Set Programming* (ASP) [5] has gradually become one of the most successful and practical formalisms for Knowledge Representation due to its flexibility, expressiveness and current availability of efficient solvers. This success can be easily checked by the continuous and plentiful presence of papers on ASP in the main conferences and journals on Logic Programming, Knowledge Representation and Artificial Intelligence during the last years. The declarative semantics of ASP has allowed many syntactic extensions that have simplified the formalisation of complex domains in different application areas like constraint satisfaction problems, planning or diagnosis.

In this paper we consider one more syntactic extension that is an underlying feature in most application domains: the use of *(partial) functions*. Most ASP

[⋆] This research was partially supported by Spanish MEC project TIN-2006-15455-C03-02 and Xunta de Galicia project INCITE08-PXIB105159PR.

programs include some predicates that are nothing else than relational representations of functions from the original domain being modelled. For instance, when modelling the typical educational example of family relationships, we may use a predicate $mother(X, Y)$ to express that X's mother is Y, but of course, we must add an additional constraint to ensure that Y is unique wrt X, i.e., that the predicate actually acts as the function $mother(X) = Y$. In fact, it is quite common that first time Prolog students use this last notation as their first attempt. Functions are not only a natural element for knowledge representation, but can also simplify in a considerable way ASP programs. Apart from avoiding constraints for uniqueness of value, the possibility of nesting functional terms like in $W = mother(father(mother(X)))$ allows a more compact and readable representation than the relational version $mother(X, Y), father(Y, Z), mother(Z, W)$ involving extra variables, which may easily mean a source of formalisation errors. Similarly, as we will see later, the use of partial functions can also save the programmer from including explicit conditions in the rule bodies to check that the rule head is actually defined.

The addition of functions to ASP is not new at all, although there exist two different ways in which functions are actually understood. Most of the existing work in the topic (like the general approaches [6–8] or the older use of function *Result* for Situation Calculus inside ASP [9]) treat functions in the same way as Prolog, that is, they are just a way for *constructing* the Herbrand universe, and so they satisfy the unique names assumption – e.g. $mother(john) = mary$ is always false. A different alternative is dealing with functions in a more similar way to Predicate Calculus, as done for instance in *Functional Logic Programming* [10]. The first and most general approach in this direction is due to the logical characterisation of ASP in terms of *Equilibrium Logic* [11] and, in particular, to its extension to first order theories, *Quantified Equilibrium Logic* (QEL) [2]. As a result of this characterisation, the concept of stable model is now defined for any theory from predicate calculus with equality. In fact, stable models can be alternatively described by a second-order logic operator [12] quite close to Circumscription [13], something that has been already used, for instance, to study strong equivalence for programs with variables [3]. Another alternative for ASP with (non-Herbrand) functions has been very recently presented in [14], with an apparently similar behaviour to [2].

As we will explain in the next section, we claim that the exclusive use of Herbrand functions and the currently proposed interpretation of equality in QEL with the requirement for functions to be complete (something that [14] imposes too) yield some counterintuitive results when introducing functions for knowledge representation. To solve these problems, we propose a variation of QEL that uses a similar structure to the logical characterisation [15] for functional logic programs, where we separate Herbrand functions (or constructors) from partial functions. We further show how our semantics for partial functions has a direct relation to the *Logic of Existence* (or *E*-logic) proposed by Scott [4].

The rest of the paper is organized as follows. In the next section, we informally consider some examples of knowledge representation with functions in

ASP, commenting the apparently expected behaviour and the problems that arise when using the current proposal for QEL. In Section 3, we introduce our variant called $\mathrm{QEL}^{=}_{\mathcal{F}}$. Section 4 defines some useful derived operators, many of them directly extracted from E-logic and showing the same behaviour. In Section 5 we consider a syntactic subclass of logic programs with partial functions and Herbrand constants, and show how they can be translated into (non-functional) normal logic programs afterwards. Finally, Section 6 contains a brief discussion about related work and Section 7 concludes the paper.

2 A Motivating Example

Consider the following simple scenario with a pair of rules.

Example 1. When deciding the second course of a given meal once the first course is fixed, we want to apply the following criterion: on Fridays, we repeat the first course as second one; the rest of week days, we choose $fish$ if the first was $pasta$. □

A straightforward encoding of these rules[1] into ASP would correspond to the program Π_1:

$$second(fish) \leftarrow first(pasta) \wedge \neg friday \tag{1}$$
$$second(X) \leftarrow first(X) \wedge friday \tag{2}$$
$$\bot \leftarrow first(X) \wedge first(Y) \wedge X \neq Y \tag{3}$$
$$\bot \leftarrow second(X) \wedge second(Y) \wedge X \neq Y \tag{4}$$

where the last two rules just represent that each course is unique, i.e., $first(salad)$ and $first(pasta)$ cannot be simultaneously true, for instance. In fact, these constraints immediately point out that $first$ and $second$ are 0-ary functions. A very naive attempt to use these functions for representing our example problem could be the pair of formulas Π_2:

$$second = fish \leftarrow first = pasta \wedge \neg friday \tag{5}$$
$$second = first \leftarrow friday \tag{6}$$

Of course, Π_2 is not a logic program, but it can still be given a logic programming meaning by interpreting it under Herbrand models of QEL, or the equivalent recent characterisation of stable models for first order theories [12]. Unfortunately, the behaviour of Π_2 in QEL with Herbrand models will be quite different to that of Π_1 by several reasons that can be easily foreseen. First of all, there exists now a qualitative difference between functions $first$ and $second$ with respect to $fish$ and $pasta$. For instance, while it is clear that $fish = pasta$ must be false, we should allow $second = first$ to cope with our Fridays criterion. If we deal

[1] As a difference wrt to the typical ASP notation, we use $\neg$ to represent default negation and, instead of a comma, we use $\wedge$ to separate literals in the body.

with Herbrand models or unique names assumption, the four constants would be pairwise different and (5) would be equivalent to $\bot \leftarrow \bot$, that is, a tautology, whereas (6) would become the constraint $\bot \leftarrow friday$.

Even after limiting the unique names assumption only to constants $fish$ and $pasta$, new problems arise. For instance, the approaches in [2, 12, 3, 14] deal with complete functions and the axiom of *decidable equality*:

$$x = y \vee \neg(x = y) \tag{DE}$$

This axiom is equivalent to $x = y \leftarrow \neg\neg(x = y)$ which informally implies that we always have a justification to assign any value to any function. Thus, for instance, if it is not Friday and we do not provide any information about the first course, i.e., no atom $first(X)$ holds, then Π_1 will not derive any information about the second course, that is, no atom $second(X)$ is derived. In Π_2, however, functions $first$ and $second$ must *always* have a value, which is further justified in any stable model by (DE). As a result, we get that a possible stable model is, for instance, $first = fish$ and $second = pasta$. A related problem of axiom (DE) is that it allows rewriting a rule like (5) as the constraint:

$$\bot \leftarrow first = pasta \wedge \neg friday \wedge \neg(second = fish)$$

whose relational counterpart would be

$$\bot \leftarrow first(pasta) \wedge \neg friday \wedge \neg second(fish) \tag{7}$$

and whose behaviour in logic programming is very different from the original rule (1). As an example, while $\Pi_1 \cup \{first(pasta)\}$ entails $second(fish)$, the same program after replacing (1) by (7) has no stable models.

Finally, even after removing decidable equality, we face a new problem that has to do with directionality in the equality symbol when used in the rule heads. The symmetry of '=' allows rewriting (6) as:

$$first = second \leftarrow friday \tag{8}$$

that in a relational notation would be the rule:

$$first(X) \leftarrow second(X) \wedge friday \tag{9}$$

which, again, has a very different meaning from the original (2). For instance $\Pi_1 \cup \{friday, second(fish)\}$ does not entail anything about the first course, whereas if we replace in this program (2) by (9), we obtain $first(fish)$. This is counterintuitive, since our program was intended to derive facts about the second course, and not about the first one. To sum up, we will need some kind of new directional operator to specify the function value in a rule head.

3 Quantified Equilibrium Logic with Partial Functions

The definition of propositional Equilibrium Logic [11] relied on establishing a selection criterion on models of the intermediate logic, called the logic of *Here-and-There* (HT) [16]. The first order case [2] followed similar steps, introducing a quantified version of HT, called SQHT$^=$ that stands for *Quantified HT with static domains*[2] *and equality*. In this section we describe the syntax and semantics of a variant, called SQHT$^=_\mathcal{F}$, for dealing with partial functions.

We begin by defining a first-order language by its *signature*, a tuple $\Sigma = \langle \mathcal{C}, \mathcal{F}, \mathcal{P} \rangle$ of disjoint sets where $\mathcal{C}$ and $\mathcal{F}$ are sets of *function names* and $\mathcal{P}$ a set of *predicate names*. We assume that each function (resp. predicate) name has the form f/n where f is the function (resp. predicate) symbol, and $n \geq 0$ is an integer denoting the number of arguments (or *arity*). Elements in $\mathcal{C}$ will be called *Herbrand functions* (or *constructors*), whereas elements in $\mathcal{F}$ will receive the name of *partial functions*. The sets $\mathcal{C}_0$ (Herbrand constants) and $\mathcal{F}_0$ (partial constants) respectively represent the elements of $\mathcal{C}$ and $\mathcal{F}$ with arity 0. We assume $\mathcal{C}_0$ contains at least one element.

First-order formulas are built up in the usual way, with the same syntax of classical predicate calculus with equality $=$. We assume that $\neg\varphi$ is defined as $\varphi \rightarrow \bot$ whereas $x \neq y$ just stands for $\neg(x = y)$. Given any set of functions $\mathcal{A}$ we write $Terms(\mathcal{A})$ to stand for the set of ground terms built from functions (and constants) in $\mathcal{A}$. In particular, the set of all possible ground terms for signature $\Sigma = \langle \mathcal{C}, \mathcal{F}, \mathcal{P} \rangle$ would be $Terms(\mathcal{C} \cup \mathcal{F})$ whereas the subset $Terms(\mathcal{C})$ will be called the *Herbrand Universe* of $\mathcal{L}$. The *Herbrand Base* $HB(\mathcal{C}, \mathcal{P})$ is a set containing all atoms that can be formed with predicates in $\mathcal{P}$ and terms in the Herbrand Universe, $Terms(\mathcal{C})$.

From now on, we assume that all free variables are implicitly universally quantified. We use letters x, y, z and their capital versions to denote variables, t to denote terms, and letters c, d to denote ground terms. Boldface letters like $\mathbf{x}, \mathbf{t}, \mathbf{c}, \ldots$ represent tuples (in this case of variables, terms and ground terms, respectively). The corresponding semantics for SQHT$^=_\mathcal{F}$ is described as follows.

Definition 1 (state). *A* state *for a signature $\Sigma = \langle \mathcal{C}, \mathcal{F}, \mathcal{P} \rangle$ is a pair (σ, A) where $A \subseteq HB(\mathcal{C}, \mathcal{P})$ is a set of atoms from the Herbrand Base and $\sigma : Terms(\mathcal{C} \cup \mathcal{F}) \rightarrow Terms(\mathcal{C}) \cup \{\mathtt{u}\}$ is a function assigning to any ground term in the language some ground term in the Herbrand Universe or the special value $\mathtt{u} \notin Terms(\mathcal{C} \cup \mathcal{F})$ (standing for* undefined*). Function σ must satisfy:*

(i) *$\sigma(c) = c$ for all $c \in Terms(\mathcal{C})$.*

(ii) $\sigma(f(t_1, \ldots, t_n)) = \begin{cases} \mathtt{u} & \textit{if } \sigma(t_i) = \mathtt{u} \textit{ for some } i = 1 \ldots n \\ \sigma(f(\sigma(t_1), \ldots, \sigma(t_n))) & \textit{otherwise} \end{cases}$

□

As we can see, our domain is exclusively formed by the terms from the Herbrand Universe, $Terms(\mathcal{C})$. These elements are used as arguments of ground atoms in the set A, that collects the *true* atoms in the state. Similarly, the value of any functional term is an element from $Terms(\mathcal{C})$, excepting the cases in which partial functions are left undefined – if so, they are assigned the special element $\mathtt{u}$ (different from any syntactic symbol) instead. Condition (i) asserts, as

[2] The term *static domain* refers to the fact that the universe is shared among all worlds in the Kripke frame.

expected, that any term c from the Herbrand Universe has the fixed valuation $\sigma(c) = c$. Condition (ii) guarantees, on the one hand, that a functional term with an undefined argument becomes undefined in its turn, and on the other hand, that functions preserve their interpretation through subterms – for instance, if we have $\sigma(f(a)) = c$ we expect that $\sigma(g(f(a))$ and $\sigma(g(c))$ coincide. It is easy to see that (ii) implies that σ is completely determined by the values it assigns to all terms like $f(\mathbf{c})$ where f is any partial function and $\mathbf{c}$ a tuple of elements in $Terms(\mathcal{C})$.

Definition 2 (Ordering $\preceq$ among states). *We say that state $S = (\sigma, A)$ is* smaller *than state $S' = (\sigma', A')$, written $S \preceq S'$, when both:*

i) $A \subseteq A'$.
ii) $\sigma(d) = \sigma'(d)$ *or* $\sigma(d) = \mathtt{u}$, *for all* $d \in Terms(\mathcal{C} \cup \mathcal{F})$. □

We write $S \prec S'$ when the relation is strict, that is, $S \preceq S'$ and $S \neq S'$. The intuitive meaning of $S \preceq S'$ is that the former contains *less information* than the latter, so that any true atom or defined function value in S must hold in S'.

Definition 3 (*HT*-interpretation). *An HT* interpretation *I for a signature $\Sigma = \langle \mathcal{C}, \mathcal{F}, \mathcal{P} \rangle$ is a pair of states $I = \langle S^h, S^t \rangle$ with $S^h \preceq S^t$.* □

The superindices h, t represent two worlds (respectively standing for *here* and *there*) with a reflexive ordering relation further satisfying $h \leq t$. An interpretation like $\langle S^t, S^t \rangle$ is said to be *total*, referring to the fact that both states contain the same information[3].

Given an interpretation $I = \langle S^h, S^t \rangle$, with $S^h = (\sigma^h, I^h)$ and $S^t = (\sigma^t, I^t)$, we define when I *satisfies* a formula φ at some world $w \in \{h, t\}$, written $I, w \models \varphi$, inductively as follows:

– $I, w \models p(t_1, \dots, t_n)$ if $p(\sigma^w(t_1), \dots, \sigma^w(t_n)) \in I^w$;
– $I, w \models t_1 = t_2$ if $\sigma^w(t_1) = \sigma^w(t_2) \neq \mathtt{u}$;
– $I, w \not\models \bot$; $I, w \models \top$;
– $I, w \models \alpha \wedge \beta$ if $I, w \models \alpha$ and $I, w \models \beta$;
– $I, w \models \alpha \vee \beta$ if $I, w \models \alpha$ or $I, w \models \beta$;
– $I, w \models \alpha \rightarrow \beta$ if for all w' s.t. $w \leq w'$: $I, w' \not\models \alpha$ or $I, w' \models \beta$;
– $I, w \models \forall x\ \alpha(x)$ if for each $c \in Terms(\mathcal{C})$: $I, w \models \alpha(c)$;
– $I, w \models \exists x\ \alpha(x)$ if for some $c \in Terms(\mathcal{C})$: $I, w \models \alpha(c)$. □

An important observation is that the first condition above implies that an atom with an undefined argument will always be valuated as false since, by definition, $\mathtt{u}$ never occurs in ground atoms of I^h or I^t. Something similar happens with equality: $t_1 = t_2$ will be false if any of the two operands, or even both, are undefined. As usual, we say that I is a *model* of a formula φ, written $I \models \varphi$, when $I, h \models \varphi$. Similarly, I is a *model* of a theory Γ when it is a model of all of its formulas. The next definition introduces the idea of equilibrium models for $\mathrm{SQHT}^{=}_{\mathcal{F}}$.

[3] Note that by *total* we do not mean that functions cannot be left undefined. We may still have some term d for which $\sigma^t(d) = \mathtt{u}$.

Definition 4 (Equilibrium model). *A model $\langle S^t, S^t\rangle$ of a theory Γ is an* equilibrium model *if there is no strictly smaller state $S^h \prec S^t$ that $\langle S^h, S^t\rangle$ is also model of Γ.* □

The Quantified Equilibrium Logic with partial functions ($\text{QEL}^{=}_{\mathcal{F}}$) is the logic induced by the $\text{SQHT}^{=}_{\mathcal{F}}$ equilibrium models.

For space reasons we describe $\text{SQHT}^{=}$ (resp. QEL) as a particular instance of $\text{SQHT}^{=}_{\mathcal{F}}$ (resp. $\text{QEL}^{=}_{\mathcal{F}}$). It can be easily checked that this description is equivalent to the one in [3]. The syntax for $\text{SQHT}^{=}$ is the same as for $\text{SQHT}^{=}_{\mathcal{F}}$ (that is, Predicate Calculus with equality) but starting from a signature $\langle \mathcal{F}, \mathcal{P}\rangle$ where no distinction is made among functions in set $\mathcal{F}$. Each $\text{SQHT}^{=}$ interpretation for signature $\langle \mathcal{F}, \mathcal{P}\rangle$ further deals with a universe domain, a set $U \neq \emptyset$ which is said to be *static*, that is, common to both worlds h and t. To capture this in $\text{SQHT}^{=}_{\mathcal{F}}$ we can just use signature $\langle \mathcal{C}, \mathcal{F}, \mathcal{P}\rangle$ and define $\mathcal{C}$ as a set of constant names, one c' per each individual $c \in U$. The most important feature of $\text{SQHT}^{=}$ interpretations is that they satisfy the axiom $t = t$ for any term t. In other words, any ground term $d \in \mathit{Terms}(\mathcal{C} \cup \mathcal{F})$ is defined $\sigma^h(d) \neq \mathtt{u}$ and, in fact, by construction of interpretations, this also means $\sigma^h(d) = \sigma^t(d)$. As a result, $\text{SQHT}^{=}$ actually uses a unique σ function for both worlds h and t and interpretations can be represented instead as $\langle \sigma, I^h, I^t\rangle$. Under this restriction, it is easy to see that decidable equality $t_1 = t_2 \vee t_1 \neq t_2$ is a valid formula.

Herbrand models from $\text{SQHT}^{=}$ and signature $\langle \mathcal{C}, \mathcal{P}\rangle$ can be easily captured by just considering $\text{SQHT}^{=}_{\mathcal{F}}$ interpretations for signature $\langle \mathcal{C}, \emptyset, \mathcal{P}\rangle$. Finally, the models selection criterion in the definition of equilibrium models need not be modified. Since $\sigma^h = \sigma^t = \sigma$ and all terms and defined, the $\preceq$ ordering relation among states in QEL actually amounts to a simple inclusion of sets of ground atoms.

4 Useful Derived Operators

From the $\text{SQHT}^{=}_{\mathcal{F}}$ semantics, it is easy to see that the formula $(t = t)$, usually included as an axiom for equality, is not valid in $\text{SQHT}^{=}_{\mathcal{F}}$. In fact, $I, w \models (t = t)$ iff $\sigma^w(t) \neq \mathtt{u}$, that is, term t is defined. In this way, we can introduce Scott's [4] *existence* operator[4] in a standard way: $E\ t \overset{\text{def}}{=} (t = t)$. Condition (ii) in Definition 1 implies the *strictness* condition of E-logic, formulated by the axiom $E\ f(t) \rightarrow E\ t$. As happens with $(t = t)$, the substitution axiom for functions:

$$t_1 = t_2 \rightarrow f(t_1) = f(t_2)$$

is not valid, since it may be the case that the function is undefined. However, the following weaker version is an $\text{SQHT}^{=}_{\mathcal{F}}$ tautology:

$$t_1 = t_2 \wedge E\ f(t_1) \rightarrow f(t_1) = f(t_2)$$

[4] Contrarily to the original Scott's E-logic, variables in $\text{SQHT}^{=}_{\mathcal{F}}$ are always defined. This is not an essential difference: terms may be left undefined instead, and so most theorems, like $(x = y) \rightarrow (y = x)$ are expressed here using metavariables for terms $(t_1 = t_2) \rightarrow (t_2 = t_1)$.

Usual axioms for equality that are valid in $\text{SQHT}^{=}_{\mathcal{F}}$ are, for any predicate P:

$$t_1 = t_2 \rightarrow t_2 = t_1$$
$$t_1 = t_2 \wedge t_2 = t_3 \rightarrow t_1 = t_3$$
$$t_1 = t_2 \wedge P(t_1) \rightarrow P(t_2)$$

To represent the *difference* between two terms, we may also have several alternatives. The straightforward one is just $\neg(t_1 = t_2)$, usually abbreviated as $t_1 \neq t_2$. However, this formula can be satisfied when any of the two operands is undefined. We may sometimes want to express a stronger notion of difference that behaves as a positive formula (this is usually called *apartness* in the intuitionistic literature [17]). In our case, we are especially interested in an apartness operator $t_1 \# t_2$ where both arguments are required to be defined:

$$t_1 \# t_2 \stackrel{\text{def}}{=} E\ t_1 \wedge E\ t_2 \wedge \neg(t_1 = t_2)$$

To understand the meaning of this operator, consider the difference between $\neg(King(France) = LouisXIV)$ and $King(Spain)\#LouisXIV$. The first expression means that we cannot prove that the King of France is Louis XIV, what includes the case in which France has not a king. The second expression means that we can prove that the King of Spain (and so, such a concept exists) is not Louis XIV.

The next operator we introduce has to do with definedness of rule heads in logic programs. The inclusion of a formula in the consequent of an implication may have an undesired effect when thinking about its use as a rule head. For instance, consider the rule $visited(next(x)) \leftarrow visited(x)$ and assume we have the fact $visited(1)$ but there is no additional information about $next(1)$. We would expect that the rule above does not yield any particular effect on $next(1)$. Unfortunately, as $visited(next(1))$ must be true, the function $next(1)$ must become defined and, as a collateral effect, it will be assigned some arbitrary value, say $next(1) = 10$ so that $visited(10)$ is made true. To avoid this problem, we will use a new operator `:-` to define a different type of implication where the consequent is only forced to be true when all the functional terms that are "necessary to build" the atoms in the consequent are defined. Given a term t we define its set of *structural arguments* $Args(t)$ as follows:

- $Args(t) \stackrel{\text{def}}{=} \{t_1, \ldots, t_n\}$ if t has the form $f(t_1, \ldots, t_n)$ for any partial function $f/n \in \mathcal{F}$.
- $Args(t) \stackrel{\text{def}}{=} t$ otherwise.

We extend this definition for any atom A, so that its set of structural arguments $Args(A)$ corresponds to:

$$Args(P(t_1, \ldots, t_n)) \stackrel{\text{def}}{=} \{t_1, \ldots, t_n\}$$
$$Args(t = t') \stackrel{\text{def}}{=} Args(t) \cup Args(t')$$

In our previous example, $Args(visited(next(x))) = \{next(x)\}$. Notice that, for an equality atom $t = t'$, we do not consider $\{t, t'\}$ as arguments as we have done for the rest of predicates, but go down one level instead, considering $Args(t) \cup Args(t')$ in its turn. For instance, if A is the atom $friends(mother(x), mother(y))$, then $Args(A)$ would be $\{mother(x), mother(y)\}$, whereas for an equality atom A' like $mother(x) = mother(y)$, $Args(A') = \{x, y\}$. We define $[\varphi]$ as the result of replacing each atom A in φ by the conjunction of all $E\ t \to A$ for each $t \in Args(A)$. We can now define the new implication operator as follows $\varphi \texttt{ :- } \psi \stackrel{\text{def}}{=} \psi \to [\varphi]$. Back to the example, if we use now $visited(next(x)) \texttt{ :- } visited(x)$ we obtain, after applying the previous definitions, that it is equivalent to:

$$
\begin{aligned}
& visited(x) \to [visited(next(x))] \\
\leftrightarrow\ & visited(x) \to (E\ next(x) \to visited(next(x))) \\
\leftrightarrow\ & visited(x) \wedge E\ next(x) \to visited(next(x))
\end{aligned}
$$

Another important operator will allow us to establish a direction in a rule head assignment – remember the discussion about distinguishing between (6) and (8) in Section 2. We define this *assignment* operator as follows:

$$f(\mathbf{t}) := t' \stackrel{\text{def}}{=} E\ t' \to f(\mathbf{t}) = t'$$

Now, our Example 1 would be encoded with the pair of formulas:

$$second := fish \texttt{ :- } first = pasta \wedge \neg friday \qquad second := first \texttt{ :- } friday$$

that, after some elementary transformations, lead to:

$$
\begin{aligned}
second = fish &\leftarrow first = pasta \wedge \neg friday \\
second = first &\leftarrow E\ first \wedge friday
\end{aligned}
$$

Using these operators, a compact way to fix a default value t' for a function $f(\mathbf{t})$ would be $f(\mathbf{t}) := t' \texttt{ :- } \neg(f(\mathbf{t}) \# t')$. Finally, we introduce a nondeterministic choice assignment with the following set-like expression:

$$f(\mathbf{t}) \in \{x \mid \varphi(x)\} \tag{10}$$

where $\varphi(x)$ is a formula (called the set *condition*) that contains the free variable x. The intuitive meaning of (10) is self-explanatory. As an example, the formula $a \in \{x \mid \exists y\ Parent(x, y)\}$ means that a should take a value among those x that are parents of some y. Expression (10) is defined as the conjunction of:

$$\forall x\ (\varphi(x) \to f(\mathbf{t}) = x \vee f(\mathbf{t}) \neq x) \tag{11}$$

$$\neg \exists x\ (\varphi(x) \wedge f(\mathbf{t}) = x) \to \bot \tag{12}$$

Other typical set constructions can be defined in terms of (10):

$$
\begin{aligned}
f(\mathbf{t}) \in \{t'(\mathbf{y}) \mid \exists \mathbf{y}\ \varphi(\mathbf{y})\} &\stackrel{\text{def}}{=} f(\mathbf{t}) \in \{x \mid \exists \mathbf{y}\ (\varphi(\mathbf{y}) \wedge t'(\mathbf{y}) = x)\} \\
f(\mathbf{t}) \in \{t'_1, \dots, t'_n\} &\stackrel{\text{def}}{=} f(\mathbf{t}) \in \{x \mid t'_1 = x \vee \dots \vee t'_n = x\}
\end{aligned}
$$

It must be noticed that variable x in (10) is not free, but implicitly quantified and local to this expression. Note that $\varphi(x)$ may contain other quantified and/or free variables. For instance, observe the difference between:

$$Person(y) \rightarrow a(y) \in \{x \mid Parent(x, y)\} \qquad (13)$$

$$Person(y) \rightarrow a(y) \in \{x \mid \exists y \; Parent(x, y)\} \qquad (14)$$

In (13) we assign, per each person y, one of her parents to $a(y)$, whereas in (13) we are assigning *any* parent as, in fact, we could change the set condition to $\exists z \; Parent(x, z)$.

At a first sight, it could seem that the formula $\exists x(\varphi(x) \wedge f(\mathbf{t}) = x)$ could capture the expected meaning of $f(\mathbf{t}) \in \{x \mid \varphi(x)\}$ in a more direct way. Unfortunately, such a formula would not "pick" a value x among those that satisfy $\varphi(x)$. For instance, if we translate $a \in \{x \mid \exists y \; Parent(x, y)\}$ as $\exists x(\exists y \; Parent(x, y) \wedge a = x)$ would allow the free addition of facts for $Parent(x, y)$. Notice also that a formula like $a \in \{t\}$ is stronger than an assignment $a := t$ since when t is undefined, the former is always false, regardless the value of a (it would informally correspond to an expression like $a \in \emptyset$).

5 Logic Programs with Partial Functions

In this section we consider a subset of $\mathrm{QEL}^{=}_{\mathcal{F}}$ which corresponds to a certain kind of logic program that allow partial functions but not constructors other than a finite set of Herbrand constants $\mathcal{C} = \mathcal{C}_0$. The interest of this syntactic class is that it can be translated into ground normal logic programs, and so, equilibrium models can be computed by any of the currently available answer set provers. From now on, we assume that any function f/n with arity $n > 0$ is partial, $f/n \in \mathcal{F}$, and any constant c is a constructor, $c \in \mathcal{C}$, unless we include a declaration $c/0 \in \mathcal{F}$. As usual in logic programming notation, we use in this section capital letters to represent variables.

In what follows we will use the tag 'FLP' to refer to functional logic programming definitions, and 'LP' to talk about the more restrictive syntax of normal logic programs (without functions). An FLP-*atom* has the form[5] $p(\mathbf{t})$ or $t_1 = t_2$, where p is a predicate name, $\mathbf{t}$ a tuple of terms and t_1, t_2 a pair of terms. An FLP-*literal* is an atom A or its default negation $\neg A$. We call LP-*terms* (resp. LP-*atoms*, resp. LP-*literals*) to those not containing partial function symbols.

An FLP-*rule* is an implication α `:-` β where β (called *body*) is a conjunction of literals, and α (called *head*) has the form of one the following expressions:

– an atom $p(\mathbf{t})$;
– the truth constant $\bot$;
– an assignment $f(\mathbf{t}) := t'$ with $f \in \mathcal{F}$;
– or a *choice* like $f(\mathbf{t}) \in \{x \mid \varphi(x)\}$ with $f \in \mathcal{F}$ and $\varphi(x)$ a conjunction of literals. We call x the *choice variable* and $\varphi(x)$ the choice condition.

[5] Expressions like $t_1 \# t_2$ are left for a future work.

A *choice rule* is a rule with a *choice* head. A *functional logic program* is a set of FLP-rules. A rule is said to be *safe* when: (1) if a variable x is the term t' or one of the terms in $\mathbf{t}$, or occurs in the scope of negation, or in a choice condition (excepting the choice variable), then it also occurs in some positive literal in the body; and (2) if x is a choice variable, then it occurs in some positive literal of the choice condition $\varphi(x)$. For instance, the rules $p(f(X), Y)$ `:-` $q(Y)$ and $f \in \{Y \mid p(Y)\}$ are safe, whereas the rules $f(Z) := 0$ or $f \in \{Y \mid \neg p(Y)\}$ are not safe. A safe program is a set of safe rules. The following is an example of a program in FLP syntax:

Example 2 (Hamiltonian cycles). Let Π_2 be the FLP-program:

$$\bot \texttt{ :- } \mathit{next}(X) = \mathit{next}(Y) \wedge X \neq Y \tag{15}$$

$$\mathit{next}(X) \in \{Z \mid \mathit{arc}(X, Z)\} \texttt{ :- } \mathit{node}(X) \tag{16}$$

$$\mathit{visited}(1) \tag{17}$$

$$\mathit{visited}(\mathit{next}(X)) \texttt{ :- } \mathit{visited}(X) \tag{18}$$

$$\bot \texttt{ :- } \neg \mathit{visited}(X) \wedge \mathit{node}(X). \tag{19}$$

An LP-*rule* is such that its body exclusively contains LP-literals and its head is either $\bot$ or an LP-atom $p(\mathbf{t})$. An LP-*program* is a set of LP-rules. It is easy to see that, for LP-rules, α `:-` β is equivalent to $\beta \rightarrow \alpha$. Thus, an LP-program has the form of a (standard) normal logic program with constraints and without partial functions. The absence of partial functions guarantees that $\mathrm{QEL}^{=}_{\mathcal{F}}$ and QEL coincide for this kind of program:

Proposition 1. *$\mathit{QEL}^{=}_{\mathcal{F}}$ equilibrium models of an LP-program Π correspond to QEL equilibrium models of Π.*

Furthermore, it is also very easy to see that, for LP-programs, the definition of *safeness* we provided generalises the standard definition for normal logic programs. As a result, this means in particular that when an LP-program Π is safe, $\mathrm{QEL}^{=}_{\mathcal{F}}$ equilibrium models coincide with the set of stable models of the grounded version of Π, since QEL satisfies this property.

The translation of an FLP-program Π will be done in two steps. In a first step, we will define a QEL theory $\Gamma(\Pi)$ for a different signature and prove that it is $\mathrm{SQHT}^{=}_{\mathcal{F}}$ equivalent modulo the original signature. This theory $\Gamma(\Pi)$ is not an LP-program, but can be easily translated into an LP-program Π^* applying some simple transformations that preserve equivalence wrt equilibrium models (even in QEL). The main idea of the translation is that, for each partial function $f/n \in \mathcal{F}$ occurring in Π we will handle a predicate like $\mathit{holds\_f}(X_1, \ldots, X_n, V)$ in Π^*, or $\mathit{holds\_f}(\mathbf{X}, V)$ for short. The technique of converting a function into a predicate and shifting the function value as an extra argument is well known in Functional Logic Programming and has received the name of *flattening* [18, 19]. Obviously, once we deal with a predicate, we will need that no two different values are assigned to the same function. This can be simply captured by:

$$\bot \leftarrow \mathit{holds\_f}(\mathbf{X}, V) \wedge \mathit{holds\_f}(\mathbf{X}, W) \wedge \neg(V = W) \tag{20}$$

with variables V, W not included in $\mathbf{X}$.

Given the original signature $\Sigma = \langle \mathcal{C}, \mathcal{F}, \mathcal{P} \rangle$ for program Π, the theory $\Gamma(\Pi)$ will deal with a new signature $\Sigma^* = \langle \mathcal{C}, \emptyset, \mathcal{P}^* \rangle$ where $\mathcal{P}^*$ consists of $\mathcal{P}$ plus a new predicate $holds\_f/(n+1)$ per each partial function $f/n \in \mathcal{F}$.

Definition 5 (Correspondence of interpretations). *Given an HT interpretation $I = \langle S^h, S^t \rangle$ for signature $\Sigma = \langle \mathcal{C}, \mathcal{F}, \mathcal{P} \rangle$ we define a* corresponding interpretation $I^* = \langle (\sigma^h, J^h), (\sigma^t, J^t) \rangle$ *for signature $\Sigma^* = \langle \mathcal{C}, \emptyset, \mathcal{P}^* \rangle$ so that, for any $f/n \in \mathcal{F}$, any tuple $\mathbf{c}$ of n elements from $\mathcal{C}$, any predicate $p/n \in \mathcal{P}$ and any $w \in \{h, t\}$:*

1. *$holds\_f(\mathbf{c}, d) \in J^w$ iff $\sigma^w(\mathbf{c}) = d$ with $d \in \mathcal{C}$.*
2. *$p(\mathbf{c}) \in J^w$ iff $p(\mathbf{c}) \in I^w$.* □

Once (20) is fixed, the correspondence between I and I^* is bidirectional:

Proposition 2. *Given signature $\Sigma = \langle \mathcal{C}, \mathcal{F}, \mathcal{P} \rangle$ and an interpretation J for Σ^* satisfying* (20)*, then there exists an interpretation I for Σ such that $I^* = J$.*

Definition 6 (Translation of terms). *We define the translation of a term t as the triple $\langle t^*, \Phi(t) \rangle$ where t^* is an LP-term and $\Phi(t)$ is a formula s.t.:*

1. *For an LP-term t, then $t^* \overset{\text{def}}{=} t$ and $\Phi(t) \overset{\text{def}}{=} \top$.*
2. *When $t = f(\mathbf{t})$ with f a partial function, then $t^* \overset{\text{def}}{=} X_t$ and $\Phi(t) \overset{\text{def}}{=} \Phi(\mathbf{t}) \wedge holds\_f(\mathbf{t}^*, X_t)$ where X_t is a new fresh variable and $\Phi(\mathbf{t})$ stands for the conjunction of all $\Phi(t_i)$ for all terms t_i in the tuple $\mathbf{t}$.* □

For 0-ary partial functions, we would have that $\mathbf{t}$ is empty – in this case we just assume that $\Phi(\mathbf{t}) = \top$. We introduce now some additional notation. Given a term t, *subterms*(t) denotes all its subterms, including t itself. Given a set of terms T, by T^* we mean $\{t^* \mid t \in S\}$. If ρ is a replacement of variables by Herbrand constants $[\mathbf{X} \leftarrow \mathbf{c}]$, we write $I, w, \rho \models \varphi$ to stand for $I, w \models \varphi[\mathbf{X} \leftarrow \mathbf{c}]$. Given a conjunction of literals $B = L_1 \wedge \dots \wedge L_n$, we denote $B^* \overset{\text{def}}{=} L_1^* \wedge \dots \wedge L_n^*$.

Definition 7 (Translation of literals). *The* translation of an atom *(or positive literal) A is a formula A^* defined as follows:*

1. *If $A = p(\mathbf{t})$, then $A^* \overset{\text{def}}{=} \exists \mathbf{X} \big(\, p(\mathbf{t}^*) \wedge \Phi(\mathbf{t}) \, \big)$ where $\mathbf{X}$ is the set of new fresh variables in subterms$(\mathbf{t})^*$ (those not occurring in the original literal).*
2. *If $A = (t_1 = t_2)$, then $A^* \overset{\text{def}}{=} \exists \mathbf{X} \big(\, t_1^* = t_2^* \wedge \Phi(t_1) \wedge \Phi(t_2) \, \big)$ where $\mathbf{X}$ is the set of new fresh variables in subterms$(t_1)^* \cup$ subterms$(t_1)^*$.*

The translation of a negative literal *$L = \neg A$ is the formula $L^* \overset{\text{def}}{=} \neg A^*$.* □

Definition 8 (Translation of rules). *The* translation of an (FLP) rule *r like H `:-` B is a conjunction of formulas $\Gamma(r)$ defined as follows:*

1. *If $H = \bot$, then $\Gamma(r)$ is the formula $\bot \leftarrow B^*$.*

2. *If H is like $p(\mathbf{t})$ then $\Gamma(r)$ is the formula $p(\mathbf{t}^*) \leftarrow \Phi(\mathbf{t}) \wedge B^*$*
3. *If H has the form $f(\mathbf{t}) := t'$ then $\Gamma(r)$ is the formula $holds\_f(\mathbf{t}^*, t'^*) \leftarrow \Phi(\mathbf{t}) \wedge \Phi(t') \wedge B^*$*
4. *If H has the form $f(\mathbf{t}) \in \{X \mid \varphi(X)\}$ then $\Gamma(r)$ is the conjunction of:*

$$holds\_f(\mathbf{t}^*, X) \vee \neg holds\_f(\mathbf{t}^*, X) \leftarrow \Phi(\mathbf{t}) \wedge B^* \wedge \varphi(X)^* \tag{21}$$

$$\bot \leftarrow \neg \exists X (holds\_f(\mathbf{t}^*, X) \wedge \varphi(X)^*) \wedge \Phi(\mathbf{t}) \wedge B^* \tag{22}$$

where we assume that, if X happened to occur in B, we have previously replaced it in the choice by a new fresh variable symbol, say $\{Y \mid \varphi(Y)\}$.

Definition 9 (Translation of a program $\Gamma(\Pi)$). *The* translation of an FLP program *Π is a theory $\Gamma(\Pi)$ consisting of the union of all $\Gamma(r)$ per each rule $r \in \Pi$ plus, for each partial function f/n, the schemata (20).* □

Theorem 1 (Correctness of $\Gamma(\Pi)$). *For any FLP-program Π with signature $\Sigma = \langle \mathcal{C}, \mathcal{F}, \mathcal{P} \rangle$ any pair of interpretations I for Σ and J for Σ^* such that $J = I^*$: $I, w \models \Pi$ iff $I^*, w \models \Gamma(\Pi)$.* □

As an example, the translation of Π_2 is the theory $\Gamma(\Pi_2)$:

$$\bot \leftarrow holds\_next(X, X_0) \wedge holds\_next(Y, X_1) \wedge X_0 = X_1 \wedge \neg(X = Y) \tag{23}$$

$$holds\_next(X, Z) \vee \neg holds\_next(X, Z) \leftarrow arc(X, Z) \wedge node(X) \tag{24}$$

$$\bot \leftarrow \neg \exists Z (holds\_next(X, Z) \wedge arc(X, Z)) \wedge node(X) \tag{25}$$

$$visited(1) \tag{26}$$

$$visited(X_2) \leftarrow holds\_next(X, X_2) \wedge visited(X) \tag{27}$$

$$\bot \leftarrow \neg visited(X) \wedge node(X) \tag{28}$$

$$\bot \leftarrow holds\_next(X, V) \wedge holds\_next(X, W) \wedge \neg(V = W) \tag{29}$$

Of course, $\Gamma(\Pi)$ is not a normal logic program, since it contains disjunction and negation in the head of "rules", whereas it may also contain expressions like $\exists X(\varphi(X))$ with $\varphi(X)$ a conjunction of literals. However, we can build an LP-program Π^* by removing these constructions and introducing new auxiliary predicates. For instance, a formula like $p \vee \neg p \leftarrow \alpha$ is equivalent (w.r.t. equilibrium models) to the pair of rules $(p \leftarrow \neg aux \wedge \alpha)$ and $(aux \leftarrow \neg p \wedge \alpha)$ where aux is a new auxiliary predicate. Similarly, we can replace a formula $\exists X(\varphi(X))$ in a rule body by a new auxiliary predicate aux', and include a rule $(aux' \leftarrow \varphi(X))$ for its definition. In our example, these transformations would replace (24) by:

$$holds\_next(X, Z) \leftarrow \neg aux(X, Z) \wedge arc(X, Z) \wedge node(X)$$
$$aux(X, Z) \leftarrow \neg holds\_next(X, Z) \wedge arc(X, Z) \wedge node(X)$$

and (25) by the rules:

$$aux'(X) \leftarrow holds\_next(X, Z) \wedge arc(X, Z) \wedge node(X)$$
$$\bot \leftarrow \neg aux'(X) \wedge node(X)$$

where, of course, the auxiliary predicates must incorporate as arguments all the free variables of the original expression they replace.

Proposition 3. *If Π is safe then Π^* is safe.* □

6 Related Work

The present approach has incorporated many of the ideas previously presented in [20, 21]. With respect to other logical characterisations of Functional Programming languages, the closest one is [15], from where we extracted the separation of constructors and partial functions. The main difference is that $\text{QEL}^{=}_{\mathcal{F}}$ provides a completely logical description of all operators that allows an arbitrary syntax (including rules with negation, disjunction in the head, etc).

Scott's E-Logic is not the only choice for logical treatment of partial functions. A related approach is the so-called *Logic of Partial Functions* (LPF) [22]. The main difference is that LPF is a three-valued logic – formulas containing undefined terms have a third, undefined truth value. The relation to (relational) ASP in this way in much more distant than the current approach, since stable models and their logical counterpart, equilibrium models, are two-valued[6].

As for the relation to other approaches exclusively dealing with Herbrand functions [6–8] it seems that they should be embedable in $\text{QEL}^{=}$, which corresponds to the fragment of $\text{QEL}^{=}_{\mathcal{F}}$ without partial functions. In a similar way, the recent approach in [14] seems to correspond to the fragment of $\text{QEL}^{=}_{\mathcal{F}}$ with complete functions (that is, the addition of decidable equality). Formal comparisons are left for future work.

7 Conclusions

This paper has tried to clarify some relevant aspects related to the use of functions in ASP for Knowledge Representation. These aspects include definedness, the treatment of equality or the directionality in function assignments. The functional nature of some predicates is hidden in many ASP domains. When functions are represented in a relational way, we require the continuous addition of constraints for uniqueness of value, and a considerable amount of extra variables to replace the ability of nesting functional terms. All this additional effort may easily become a source for programming errors. Although, as we have shown, the proposed approach can be translated into relational ASP and merely considered as *syntactic sugar*, we claim that the use of functions may provide a more natural, compact and readable way of representing many scenarios. The previous experience with a very close language to that of Section 5, implemented in an online interpreter[7] and used for didactic purposes in the past, shows that the functional notation helps the student concentrate on the mathematical definition of the domain to be represented, and forget some of the low level representation tasks, as those commented above, or as the definedness conditions, that must be also considered in the relational representation, but the functional interpreter checks in an automatic way.

We hope that the current approach will help to integrate, in the future, the explicit treatment of arithmetic functions made by some ASP tools, that are

[6] Note that in this work we are not considering explicit negation.

[7] Available at `http://www.dc.fi.udc.es/~cabalar/fal/`

currently handled *outside* the formal setting. For instance, the ASP grounder `lparse`[8] syntactically accepts a program like $p(div(10, X)) \leftarrow q(X)$ but raises a "divide by zero" runtime error if fact $q(0)$ is added to the program. On the other hand, when div is replaced by a non-built-in function symbol, say f, the meaning is quite different, and we get $\{p(f(10, 0)), q(0)\}$ as a stable model. In this paper we have also identified and separated evaluable and (possibly) partial functions (like div above) from constructors (like f in the previous example).

We have provided a translation of our functional language into normal logic programs to show that: (1) it can be implemented with current ASP solvers; but more important (2) that the proposed semantics is *sensible* with respect to the way in which we usually program in the existing ASP paradigm. A topic for future study is the implementation of a solver that directly handles the functional semantics. Other open topics are the axiomatisation of the current logical framework or the addition of a second, explicit (or strong) negation.

Acknowledgements I am especially thankful to Joohyung Lee and Yunsong Meng for pointing out some technical errors in a preliminary version of this work, and to the anonymous referees for their helpful suggestions.

References

1. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Proc. of the 5th Intl. Conf. on Logic Programming. (1988) 1070–1080
2. Pearce, D., Valverde, A.: Towards a first order equilibrium logic for nonmonotonic reasoning. In: Proc. of the 9th European Conf. on Logics in AI (JELIA'04). (2004) 147–160
3. Lifschitz, V., Pearce, D., Valverde, A.: A characterization of strong equivalence for logic programs with variables. In: Proc. of the 9th Intl. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'07). (2007) 188–200
4. Scott, D.: Identity and existence in intuitionistic logic. Lecture Notes in Mathematics **753** (1979) 660–696
5. Marek, V., Truszczyński, M. In: Stable models and an alternative logic programming paradigm. Springer-Verlag (1999) 169–181
6. Syrjänen, T.: Omega-restricted logic programs. In: Proc. of the 6th Intl. Conf. on Logic Programming and Nonmonotonic Reasoning (LNCS 2173). (2001) 267–279
7. Bonatti, P.A.: Reasoning with infinite stable models. Artificial Intelligence **156** (2004) 75–111
8. Šimkus, M., Eiter, T.: Decidable non-monotonic disjunctive logic programs with function symbols. In: Proc. of the 14th Intl. Conf. on Logic for Programming, Artificial Intelligence (LNCS 4790). (2007) 514–530
9. Gelfond, M., Lifschitz, V.: Representing action and change by logic programs. Journal of Logic Programming **17** (1993) 301–321
10. Hanus, M.: The integration of functions into logic programming: from theory to practice. Journal of Logic Programming **19,20** (1994) 583–628
11. Pearce, D.: A new logical characterisation of stable models and answer sets. In: Non monotonic extensions of logic programming. Proc. NMELP'96. (LNAI 1216). Springer-Verlag (1996)

[8] Available at `http://www.tcs.hut.fi/Software/smodels/`.

12. Ferraris, P., Lee, J., Lifschitz, V.: A new perspective on stable models. In: Proc. of the International Joint Conference on Artificial Intelligence (IJCAI'07). (2004) 372–379
13. McCarthy, J.: Circumscription: A form of non-monotonic reasoning. Artificial Intelligence **13** (1980) 27–39
14. Lin, F., Wang, Y.: Answer set programming with functions. In: Proc. of the 11th Intl. Conf. on Principles of Knowledge Representation and Reasoning (KR'08) (to appear). (2008)
15. Almendros-Jiménez, J.M., Gavilanes-Franco, A., Gil-Luezas, A.: Algebraic semantics for functional logic programming with polymorphic order-sorted types. In: Proc. of the 5th Intl. Conf. on Algebraic and Logic Programming (ALP'96). (1996)
16. Heyting, A.: Die formalen Regeln der intuitionistischen Logik. Sitzungsberichte der Preussischen Akademie der Wissenschaften, Physikalisch-mathematische Klasse (1930) 42–56
17. Heyting, A.: Intuitionism. An Introduction. North-Holland (1956)
18. Naish, L.: Adding equations to NU-Prolog. In: Proc. of the 3rd Intl. Symp. on Programming Language Implementation and Logic Programming. Number 528 in LNCS, Springer-Verlag (1991) 15–26
19. Rouveirol, C.: Flattening and saturation: Two representation changes for generalization. Machine Learning **14**(1) (1994) 219–232
20. Cabalar, P., Lorenzo, D.: Logic programs with functions and default values. In: Proc. of the 9th European Conf. on Logics in AI (JELIA'04) (LNCS 3229). (2004) 294–306
21. Cabalar, P.: A functional action language front-end. In: Presentation at the 3rd Workshop on Answer Set Programming (ASP'05). (2005) available at `http://www.dc.fi.udc.es/ai/~cabalar/asp05_C.pdf/`.
22. Barringer, H., Cheng, H., Jones, C.B.: A logic covering undefinedness in program proofs. Acta Informatica **21** (1984) 251–269

Appendix. Proofs

(Only for revision purposes: to be removed in the final version)

Proof (of Proposition 2). Then, it suffices with defining $I^w = \{p(\mathbf{c}) \in J^w \mid p/n \in \mathcal{P}\}$ and σ^w such that, for any partial function f and tuple $\mathbf{c}$ in elements of $Terms(\mathcal{C})$: $\sigma^w(f(\mathbf{c})) = d$ if $holds\_f(\mathbf{c}, d) \in J^w$; or $\sigma^w(f(\mathbf{c})) = \mathtt{u}$ otherwise. Note that the latter is well-defined since (20) guarantees that no pair of atoms $holds\_f(\mathbf{c}, d)$ and $holds\_f(\mathbf{c}, e)$ with $e \neq d$ are included in any J^w. The rest of mapping σ^w is built up from its structural definition implied by Condition (ii) in Definition 1. $\square$

Lemma 1. *For any term t, interpretation I and corresponding interpretation I^*, and for any replacement ρ of variables in $subterms(t)^*$ then $I^*, w, \rho \models \Phi(t)$ is equivalent to: $I, w, \rho \models E\ t$ and $I, w, \rho \models (t')^* = t'$ for any $t' \in subterms(t)$.* $\square$

Proof. We proceed by induction. For the base case, when t is an LP-term, $E\ t$ is valid, and so equivalent to $\top = \Phi(t)$; besides, $t^* = t$ by definition and t has no subterms. Assume proved for a tuple of terms $\mathbf{t}$ and consider $t = f(\mathbf{t})$. Then note that $I^*, w, \rho \models \Phi(t)$ is equivalent to condition (A): $I^*, w, \rho \models \Phi(\mathbf{t})$ and $I^*, w, \rho \models holds\_f(\mathbf{t}^*, X_t)$. Now the first conjunct of (A) is equivalent, by induction, to $I, w, \rho \models E\ \mathbf{t}$ and $I, w \models (t')^* = t'$ for any subterm of $\mathbf{t}$, whereas the second conjunct of (A) is equivalent, by the correspondence between I and I^*, to $I, w, \rho \models f(\mathbf{t}) = X_t$ provided that we have already obtained $I, w, \rho \models \mathbf{t}^* = \mathbf{t}$. To sum up, (A) is therefore equivalent to $I, w, \rho \models E\ \mathbf{t} \wedge f(\mathbf{t}) = X_t$ and $I, w, \rho \models (t')^* = t'$ for any subterm of $\mathbf{t}$. Since $E\ \mathbf{t} \wedge f(\mathbf{t}) = X_t$ is equivalent to $E\ f(\mathbf{t}) \wedge f(\mathbf{t}) = X_t$ and this, by definition, is the same than $E\ t \wedge t = t^*$, we finally obtain $I, w, \rho \models E\ t$ and $I, w, \rho \models (t')^* = t'$ for any subterm of t. $\square$

Lemma 2. *For any body literal L: $I^*, w \models L^*$ iff $I, w \models L$.* $\square$

Proof. Depending on the form of L we have:

1. If L is some atom $p(\mathbf{t})$, then $I^*, w \models L^*$ means that for some substitution ρ of variables in $subterms(\mathbf{t})^*$: $I^*, w, \rho \models p(\mathbf{t}^*)$ and $I^*, w, \rho \models \Phi(\mathbf{t})$. By Lemma 1, the second conjunct is equivalent to $I, w, \rho \models E\ \mathbf{t}$ and $I, w, \rho \models t^* = t$ for any subterm t of $\mathbf{t}$ (and so of L), and in particular $I, w, \rho \models \mathbf{t}^* = \mathbf{t}$. But this means that $I^*, w, \rho \models p(\mathbf{t}^*)$ is equivalent to $I, w, \rho \models p(\mathbf{t})$ by the correspondence of I and I^*. Since $p(\mathbf{t})$ implies $E\ \mathbf{t}$ we can remove the latter and, as a result, the original condition $I^*, w, \rho \models L^*$ is equivalent to $I, w, \rho \models p(\mathbf{t})$ and $I, w, \rho \models t^* = t$ for any subterm t of L. As $p(\mathbf{t})$ does not contain variables in $subterms(\mathbf{t})^*$, the previous conditions are equivalent to: $I, w \models p(\mathbf{t})$ and there exists some ρ for which $I, w, \rho \models t^* = t$. But as $I, w \models p(\mathbf{t})$ means that $p(\mathbf{t})$ is defined in I, w, the existence of a substitution ρ for variables in $subterms(\mathbf{t})^*$ that satisfies $I, w, \rho \models t^* = t$ for any subterm t of L is guaranteed, and so, is a redundant condition that can be removed.
2. If L has the form $t_1 = t_2$ then the proof follows similar steps to case 1.

3. If L has the form $\neg A$, then $I^*, w \models \neg A^*$ is equivalent to $I^*, t \not\models A^*$. Applying the proof for cases 1 and 2 to atom A, this is equivalent to $I, t \not\models A$ that is further equivalent to $I, w \models \neg A$. $\Box$

Obviously, Lemma 2 directly implies that $I, w \models B$ is equivalent to $I^*, w \models B^*$.

Lemma 3. *$I^*, w \models \Gamma(r)$ iff $I, w \models r$.* $\Box$

Proof. If $r = (H$ `:-` $B)$, depending on the form of H we have:

1. If $H = \bot$, is easy to see that $(\bot$ `:-` $B)$ is equivalent to $(\bot \leftarrow B)$. Then, $I^*, w \models \bot \leftarrow B^* \Leftrightarrow I^*, t \not\models B^* \Leftrightarrow$ (by Lemma 2) $I, t \not\models B \Leftrightarrow I, w \models \bot \leftarrow B$.
2. If H is like $p(\mathbf{t})$, then $p(\mathbf{t})$ `:-` B is equivalent to $p(\mathbf{t}) \leftarrow B \wedge E\ \mathbf{t}$. Then, $I^*, w \models p(\mathbf{t}^*) \leftarrow \Phi(\mathbf{t}) \wedge B^* \Leftrightarrow$ for all $w' \geq w$: if $I^*, w' \models \Phi(\mathbf{t}) \wedge B^*$ then $I^*, w' \models p(\mathbf{t}^*)$. Let us call (A) to this condition. By Lemma 2, $I^*, w' \models B^*$ is equivalent to $I, w' \models B$. Now note that rules are universally quantified. Take any replacement ρ of variables in $\mathit{subterms}(\mathbf{t})^*$. By Lemma 1, $I^*, w', \rho \models \Phi(\mathbf{t})$ is equivalent to $I, w', \rho \models E\ \mathbf{t}$ and $I, w', \rho \models t'^* = t'$ for any $t' \in \mathit{subterms}(\mathbf{t})$. If this holds, $I^*, w', \rho \models p(\mathbf{t}^*)$, which coincides with $I, w', \rho \models p(\mathbf{t}^*)$, is equivalent to $I, w', \rho \models p(\mathbf{t})$. To sum up, (A) is equivalent to: for all $w' \geq w$, if $I, w', \rho \models B \wedge E\ \mathbf{t}$ then $I, w', \rho \models p(\mathbf{t})$ for any replacement ρ. But this is the same than $I, w \models p(\mathbf{t}) \leftarrow B \wedge E\ \mathbf{t}$.
3. If H has the form $f(\mathbf{t}) := t'$, we may first observe that $(H$ `:-` $B)$ is equivalent to $f(\mathbf{t}) = t' \leftarrow E\ \mathbf{t} \wedge E\ t' \wedge B$. Then, $I^*, w \models \mathit{holds\_f}(\mathbf{t}^*, t'^*) \leftarrow \Phi(\mathbf{t}) \wedge \Phi(t') \wedge B^*$ is equivalent to, for any world $w' \geq w$ and any replacement of variables ρ: if $I^*, w', \rho \models \Phi(\mathbf{t}) \wedge \Phi(t') \wedge B^*$ then $I^*, w', \rho \models \mathit{holds\_f}(\mathbf{t}^*, t'^*)$. By Lemmas 1 and 2, the antecedent is equivalent to $I, w', \rho \models E\ \mathbf{t} \wedge E\ t' \wedge B$ plus $I, w', \rho \models k^* = k$ for each $k \in \mathit{subterms}(\mathbf{t} \cdot t')$. On the other hand, $I^*, w', \rho \models \mathit{holds\_f}(\mathbf{t}^*, t'^*)$ is equivalent, by correspondence of I and I^*, to $I, w', \rho \models f(\mathbf{t}^*) = t'^*$ and this, in presence of the equivalent condition for the antecedent we obtained before, is equivalent to $I, w', \rho \models f(\mathbf{t}) = t'$. The rest of the proof follows as in the previous case.
4. If H has the form $f(\mathbf{t}) \in \{X \mid \varphi(X)\}$ then, after some simple transformations, it can be checked that $(H$ `:-` $B)$ is equivalent to the conjunction of the formulas:

$$f(\mathbf{t}) = X \vee \neg f(\mathbf{t}) = X \leftarrow \varphi(X) \wedge E\ \mathbf{t} \wedge B \tag{30}$$

$$\bot \leftarrow \neg \exists X(\varphi(X) \wedge f(\mathbf{t}) = X) \wedge E\ \mathbf{t} \wedge B \tag{31}$$

   The proof for this case is tedious, but follows similar steps to the previous two cases. By analogy, it is not difficult to see that $I, w \models (30)$ iff $I^*, w \models (21)$ and that $I, w \models (31)$ iff $I^*, w \models (22)$. $\Box$

Proof (of Theorem 1). The proof directly follows from Lemma 3. $\Box$