Extending Answer Set Programs with Interpreted Functions as First-class Citizens

Christoph Redl

redl@kr.tuwien.ac.at

TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

Knowledge-Based Systems Group

January 16, 2017

Outline

Motivation

Function Symbols in Answer Set Programs

- Function symbols are often uninterpreted and are used for structuring information.

Motivation

Function Symbols in Answer Set Programs

- Function symbols are often uninterpreted and are used for structuring information.
  Example: $multiply(add(4,5),3)$ represents the expression $(4+5)\cdot 3$, but does not actually evaluate it.

Motivation

Function Symbols in Answer Set Programs

- Function symbols are often uninterpreted and are used for structuring information.
  Example: $multiply(add(4,5),3)$ represents the expression $(4+5)\cdot 3$, but does not actually evaluate it.
- Existing approaches towards interpreted functions typically define functions as part of the program.

Motivation

Function Symbols in Answer Set Programs

- Function symbols are often uninterpreted and are used for structuring information.
  Example: $multiply(add(4,5),3)$ represents the expression $(4+5)\cdot 3$, but does not actually evaluate it.
- Existing approaches towards interpreted functions typically define functions as part of the program.
  Example: $loc(X) = garage \leftarrow car(X), \mathrm{not}\, loc(X) \neq garage$

Motivation

Function Symbols in Answer Set Programs

- Function symbols are often uninterpreted and are used for structuring information.
  Example: $multiply(add(4, 5), 3)$ represents the expression $(4 + 5) \cdot 3$, but does not actually evaluate it.
- Existing approaches towards interpreted functions typically define functions as part of the program.
  Example: $loc(X) = garage \leftarrow car(X), \mathrm{not}\, loc(X) \neq garage$
- Externally defined semantics of function symbols are supported by only few approaches (e.g. HEX-programs, VI-programs, Clingo5).

Motivation

Function Symbols in Answer Set Programs

- Function symbols are often uninterpreted and are used for structuring information.
  Example: $multiply(add(4,5),3)$ represents the expression $(4+5)\cdot 3$, but does not actually evaluate it.
- Existing approaches towards interpreted functions typically define functions as part of the program.
  Example: $loc(X) = garage \leftarrow car(X), \text{not}\, loc(X) \neq garage$
- Externally defined semantics of function symbols are supported by only few approaches (e.g. HEX-programs, VI-programs, Clingo5).
  Example: $result(Y) \leftarrow \&add[4,5](X), \&multiply[X,3](Y)$

Motivation

Function Symbols in Answer Set Programs

- Function symbols are often uninterpreted and are used for structuring information.
  Example: $multiply(add(4,5),3)$ represents the expression $(4+5)\cdot 3$, but does not actually evaluate it.
- Existing approaches towards interpreted functions typically define functions as part of the program.
  Example: $loc(X) = garage \leftarrow car(X), \text{not}\, loc(X) \neq garage$
- Externally defined semantics of function symbols are supported by only few approaches (e.g. HEX-programs, VI-programs, Clingo5).
  Example: $result(Y) \leftarrow \&add[4,5](X), \&multiply[X,3](Y)$
  But the functions are not first-class citizens
  $\Rightarrow$ this inhibits higher-order functions.

Motivation

Function Symbols in Answer Set Programs

- Function symbols are often uninterpreted and are used for structuring information.
  Example: $multiply(add(4,5),3)$ represents the expression $(4+5)\cdot 3$, but does not actually evaluate it.
- Existing approaches towards interpreted functions typically define functions as part of the program.
  Example: $loc(X) = garage \leftarrow car(X), \text{not}\, loc(X) \neq garage$
- Externally defined semantics of function symbols are supported by only few approaches (e.g. HEX-programs, VI-programs, Clingo5).
  Example: $result(Y) \leftarrow \&add[4,5](X), \&multiply[X,3](Y)$
  But the functions are not first-class citizens
  $\Rightarrow$ this inhibits higher-order functions.

Goal: Using externally defined functions, but being able to access them as objects, compose them to new functions and pass them to other functions.

Motivation

Main idea

- Represent interpreted functions themselves by terms in the program.
- This turns them into first-class citizens, i.e., accessible objects.

Motivation

Main idea

- Represent interpreted functions themselves by terms in the program.
- This turns them into first-class citizens, i.e., accessible objects.
- Since they are objects in the program, they can be passed to other functions.

Motivation

Main idea

- Represent interpreted functions themselves by terms in the program.
- This turns them into first-class citizens, i.e., accessible objects.
- Since they are objects in the program, they can be passed to other functions.
- At specific points, they can be applied to a list of parameters.

Motivation

Main idea

- Represent interpreted functions themselves by terms in the program.
- This turns them into first-class citizens, i.e., accessible objects.
- Since they are objects in the program, they can be passed to other functions.
- At specific points, they can be applied to a list of parameters.
- This paves the way for new modeling techniques:
  abstract usage of functions, import of functions from outside, design patterns, higher-order techniques from functional programming.

Motivation

Main idea

- Represent interpreted functions themselves by terms in the program.
- This turns them into first-class citizens, i.e., accessible objects.
- Since they are objects in the program, they can be passed to other functions.
- At specific points, they can be applied to a list of parameters.
- This paves the way for new modeling techniques:
  abstract usage of functions, import of functions from outside, design patterns, higher-order techniques from functional programming.

Contribution

- Representation of functions as terms.
- Based on this representation, we present HEX^{IFU}-programs.
- A translation of such programs to traditional HEX-programs.
- Applications.

Outline

Representing Interpreted Functions by Terms

Basic functions

- Function symbols $f \in \mathcal{F}$ are basic function associated with an arity ℓ.
- We assume that each $f \in \mathcal{F}$ has an associated (total) semantics function $sem_f(\vec{y})\colon \mathcal{C}^\ell \mapsto \mathcal{T}$ defined for all ℓ-ary vectors $\vec{y} \in \mathcal{C}^\ell$ of constants
  $\mathcal{T}$ ... set of all function terms constructible over $\mathcal{F}$ and $\mathcal{C}$.

Representing Interpreted Functions by Terms

Basic functions

- Function symbols $f \in \mathcal{F}$ are basic function associated with an arity ℓ.
- We assume that each $f \in \mathcal{F}$ has an associated (total) semantics function $sem_f(\vec{y}) \colon \mathcal{C}^\ell \mapsto \mathcal{T}$ defined for all ℓ-ary vectors $\vec{y} \in \mathcal{C}^\ell$ of constants
$\mathcal{T}$ . . . set of all function terms constructible over $\mathcal{F}$ and $\mathcal{C}$.

Representing general (possibly composed) functions

- We let $\mathcal{C}$ contain constant symbols $\#i$ for all integers $i \geq 1$ (placeholders), which are used to represent function parameters.
- We use $\mathcal{T}$ as function-representing (fr-)terms to turn interpreted functions into accessible objects.

Representing Interpreted Functions by Terms

Example

Assume that the basic functions $multiply$ and add have the expected semantics.

Then the fr-term $t_1 = multiply(add(\#1, \#2), \#3)$ represents in standard mathematical notation the function $\hat{t_1}(p_1, p_2, p_3) = (p_1 + p_2) \cdot p_3$.

Representing Interpreted Functions by Terms

Example

Assume that the basic functions *multiply* and *add* have the expected semantics.

Then the fr-term $t_1 = multiply(add(\#1, \#2), \#3)$ represents in standard mathematical notation the function $\hat{t_1}(p_1, p_2, p_3) = (p_1 + p_2) \cdot p_3$.

Note

An fr-term $t = f(t_1, \ldots, t_\ell)$ with $f \in \mathcal{F}$ and $t_1, \ldots, t_\ell \in \mathcal{T}$ does not represent the application of f to $t_1, \ldots, t_\ell$, but does itself represents a (composed) function.

Representing Interpreted Functions by Terms

Definition

For a list of ground terms $t, p_1, \ldots, p_{\gamma(t)}$ we let $val(t, p_1, \ldots, p_{\gamma(t)})$ be given by

$$\begin{cases} val(sem_f(\vec{t'}), p_1, \ldots, p_{\gamma(t)}) & \text{if } t = f(\vec{t}) \text{ and } \vec{t'} \text{ is free of } \#i, \\ f(\vec{t'}) & \text{if } t = f(\vec{t}) \text{ and there is a } \#i \text{ in } \vec{t'}, \\ p_i & \text{if } t = \#i \text{ for some } 1 \leq i \leq \gamma(t), \\ t & \text{otherwise,} \end{cases}$$

where $\vec{t}$ and $\vec{t'}$ are ℓ-ary vectors with $t'_i = val(t_i, p_1, \ldots, p_{\gamma(t)})$ for all $1 \leq i \leq \ell$.

Representing Interpreted Functions by Terms

Definition

For a list of ground terms $t, p_1, \ldots, p_{\gamma(t)}$ we let $val(t, p_1, \ldots, p_{\gamma(t)})$ be given by

$$\begin{cases} val(sem_f(\vec{t'}), p_1, \ldots, p_{\gamma(t)}) & \text{if } t = f(\vec{t}) \text{ and } \vec{t'} \text{ is free of } \#i, \\ f(\vec{t'}) & \text{if } t = f(\vec{t}) \text{ and there is a } \#i \text{ in } \vec{t'}, \\ p_i & \text{if } t = \#i \text{ for some } 1 \leq i \leq \gamma(t), \\ t & \text{otherwise,} \end{cases}$$

where $\vec{t}$ and $\vec{t'}$ are ℓ-ary vectors with $t'_i = val(t_i, p_1, \ldots, p_{\gamma(t)})$ for all $1 \leq i \leq \ell$.

Example

Consider $t = multiply(\underbrace{add(\overbrace{\#1}^{t_1}, \overbrace{\#2}^{t_2})}_{t_4}, \overbrace{\#3}^{t_3})$ and suppose to evaluate $\hat{t}(4, 5, 3)$.

- $t'_1 = val(\#1, 4, 5, 3) = 4$, $t'_2 = val(\#2, 4, 5, 3) = 5$, $t'_3 = val(\#3, 4, 5, 3) = 3$
- $t'_4 = val(add(\#1, \#2), t'_1, t'_2, t'_3) = val(add(\#1, \#2), 4, 5, 3) = 9$
- $t' = val(sem_{multiply}(t'_4, t'_3)) = val(sem_{multiply}(9, 3)) = 27$

Using FR-Terms in Programs

Definition

An interpreted function (ifu-)atom is of kind

$$\bar{R} =_{\$} \bar{T}[\bar{P}_1, \ldots, \bar{P}_\ell],$$

where $\bar{R} \in \mathcal{T}$ is a comparison operand, $\bar{T} \in \mathcal{T}$ is an fr-term, and $\bar{P}_1, \ldots, \bar{P}_\ell \in \mathcal{T}$ are parameters.

Using FR-Terms in Programs

Definition

An interpreted function (ifu-)atom is of kind

$$\bar{R} =_{\$} \bar{T}[\bar{P}_1, \dots, \bar{P}_\ell],$$

where $\bar{R} \in \mathcal{T}$ is a comparison operand, $\bar{T} \in \mathcal{T}$ is an fr-term, and $\bar{P}_1, \dots, \bar{P}_\ell \in \mathcal{T}$ are parameters.

Definition

A ground ifu-atom a of form $r =_{\$} t[t_1, \dots, t_n]$ is true wrt. assignment A, denoted $A \models a$, if $n = \gamma(t)$ and r has the value of $val(t, t_1, \dots, t_n)$, and false, denoted $A \not\models a$, otherwise.

Using FR-Terms in Programs

Definition

An interpreted function (ifu-)atom is of kind

$$\bar{R} =_{\$} \bar{T}[\bar{P}_1, \ldots, \bar{P}_\ell],$$

where $\bar{R} \in \mathcal{T}$ is a comparison operand, $\bar{T} \in \mathcal{T}$ is an fr-term, and $\bar{P}_1, \ldots, \bar{P}_\ell \in \mathcal{T}$ are parameters.

Definition

A ground ifu-atom a of form $r =_{\$} t[t_1, \ldots, t_n]$ is true wrt. assignment A, denoted $A \models a$, if $n = \gamma(t)$ and r has the value of $val(t, t_1, \ldots, t_n)$, and false, denoted $A \not\models a$, otherwise.

Example

The fr-term $add(\#1, 1)$ represents the increment function.
The ifu-atom $X =_{\$} add(\#1, 1)[Y]$ applies it to the parameter Y and compares the result with X.

Using FR-Terms in Programs

Definition

An ASP- resp. HEX-program with interpreted functions (ASP^{IFU} resp. HEX^{IFU}) is an ASP- resp. HEX-program, where rule bodies may contain ifu-atoms.

Using FR-Terms in Programs

Definition

An ASP- resp. HEX-program with interpreted functions (ASP$^{\text{IFU}}$resp. HEX$^{\text{IFU}}$) is an ASP- resp. HEX-program, where rule bodies may contain ifu-atoms.

Example

- Consider
  $compInitials(\quad concat(firstchar(\#1), firstchar(\#2)) \quad) \leftarrow$
- Consider facts of kind
  $person(F, L) \leftarrow$
  represent persons with first name F and last name L.

Using FR-Terms in Programs

Definition

An ASP- resp. HEX-program with interpreted functions (ASP$^{\mathsf{IFU}}$resp. HEX$^{\mathsf{IFU}}$) is an ASP- resp. HEX-program, where rule bodies may contain ifu-atoms.

Example

- Consider
  $compInitials(\quad concat(firstchar(\#1), firstchar(\#2)) \quad) \leftarrow$
- Consider facts of kind
  $person(F, L) \leftarrow$
  represent persons with first name F and last name L.
- Then
  $initials(F, L, I) \leftarrow person(F, L), compInitials(C), I =_{\$} C[F, L]$
  computes the initials of all persons by applying the function.

Outline

HEX-Programs

HEX-programs extend ordinary ASP programs by external sources

Definition (HEX-programs)

A HEX-program consists of rules of form

$$a_1 \vee \cdots \vee a_n \leftarrow b_1, \ldots, b_m, \text{not } b_{m+1}, \ldots, \text{not } b_n,$$

with classical literals a_i, and classical literals or an external atoms b_j.

Definition (External Atoms)

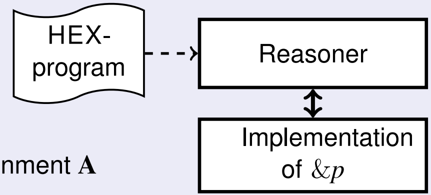
An external atom is of the form

$$\&p[q_1, \ldots, q_k](t_1, \ldots, t_l),$$

p ... external predicate name

q_i ... predicate names or constants

t_j ... terms



Semantics:

$1 + k + l$-ary Boolean oracle function $f_{\&p}$:

$\&p[q_1, \ldots, q_k](t_1, \ldots, t_l)$ is true under assignment $\mathbf{A}$

iff $f_{\&p}(\mathbf{A}, q_1, \ldots, q_k, t_1, \ldots, t_l) = 1$.

Outline

Evaluation of HEX$^{\mathsf{IFU}}$-Programs

Evaluation is based on a translation to traditional HEX-programs.

Definition

For an assignment A and list of ground terms $t, p_1, \ldots, p_n$ s.t. $\gamma(t) = n$, let $f_{\&eval}(A, t, p_1, \ldots, p_n, r) = \sigma$ where $\sigma = \mathbf{T}$ if $r = val(t, p_1, \ldots, p_n)$ and $\sigma = \mathbf{F}$ otherwise.

Evaluation of HEX$^{\text{IFU}}$-Programs

Evaluation is based on a translation to traditional HEX-programs.

Definition

For an assignment A and list of ground terms $t, p_1, \ldots, p_n$ s.t. $\gamma(t) = n$, let $f_{\&eval}(A, t, p_1, \ldots, p_n, r) = \sigma$ where $\sigma = \mathbf{T}$ if $r = val(t, p_1, \ldots, p_n)$ and $\sigma = \mathbf{F}$ otherwise.

Definition

The translation of an ifu-atom a of kind $\bar{R} =_{\$} \bar{T}[\bar{P}_1, \ldots, \bar{P}_\ell]$ to an external atom is given by $\tau(a) = \&eval[\bar{T}, \bar{P}_1, \ldots, \bar{P}_\ell](\bar{R})$.

For HEX$^{\text{IFU}}$-program Π, we let $\tau(\Pi)$ be Π after replacing each ifu-atom a by $\tau(a)$.

Evaluation of HEX$^{\text{IFU}}$-Programs

Evaluation is based on a translation to traditional HEX-programs.

Definition

For an assignment A and list of ground terms $t, p_1, \ldots, p_n$ s.t. $\gamma(t) = n$, let $f_{\&eval}(A, t, p_1, \ldots, p_n, r) = \sigma$ where $\sigma = \mathbf{T}$ if $r = val(t, p_1, \ldots, p_n)$ and $\sigma = \mathbf{F}$ otherwise.

Definition

The translation of an ifu-atom a of kind $\bar{R} =_{\$} \bar{T}[\bar{P}_1, \ldots, \bar{P}_\ell]$ to an external atom is given by $\tau(a) = \&eval[\bar{T}, \bar{P}_1, \ldots, \bar{P}_\ell](\bar{R})$.

For HEX$^{\text{IFU}}$-program Π, we let $\tau(\Pi)$ be Π after replacing each ifu-atom a by $\tau(a)$.

Proposition

An assignment A is an answer set of a HEX$^{\text{IFU}}$*-program Π if and only if it is an answer set of the* HEX*-program $\tau(\Pi)$.*

Outline

Applications of HEX$^{\text{IFU}}$-Programs

Software design patterns

- Consider $\&getValidator[type](V)$ which returns for a given type of data $type \in \{phone, email, url, \ldots\}$ a validator.

Applications of HEX$^{\text{IFU}}$-Programs

Software design patterns

- Consider $\&getValidator[type](V)$ which returns for a given type of data $type \in \{phone, email, url, \dots\}$ a validator.
- Data can be verified using:

$$
\begin{aligned}
r_1: validators(AttType, V) &\leftarrow emp(Id, AttType, AttValue),\\
&\quad \&getValidator[AttType](V).\\
r_2: invalid(Id) &\leftarrow emp(Id, AttType, AttValue),\\
&\quad validators(AttType, V), 0 =_{\$} V[AttValue].
\end{aligned}
$$

Applications of HEX$^{\text{IFU}}$-Programs

Integration of heterogeneous knowledge bases

Suppose $lookup(\#1)$ provides access to the central dictionary and is accessible via predicate l.

Then $data(A) \leftarrow l(D), K =_{\$} D[employee], A =_{\$} K[query]$ can be used to answer queries over the *employee* knowledge-base using the access function D.

Applications of HEX$^{\text{IFU}}$-Programs

Integration of heterogeneous knowledge bases

Suppose $lookup(\#1)$ provides access to the central dictionary and is accessible via predicate l.

Then $data(A) \leftarrow l(D), K =_{\$} D[employee], A =_{\$} K[query]$ can be used to answer queries over the $employee$ knowledge-base using the access function D.

Traditional higher-order functions

Consider the external atom $\&map[f,p](X)$ which applies function f, given as an fr-term, to all elements in the extension of predicate p.

Then $res(R) \leftarrow compInitials(C), R =_{\$} \&map[C, person](X)$ can be used to compute the initials of all persons in the extension of predicate $person$.

Outline

Conclusion

ASP Programs with Interpreted Functions

- Traditionally, functions are mostly
  either uninterpreted or interpreted but defined within the program.
- Our approach uses externally defined functions.
- In contrast to few existing approaches towards such externally defined functions, ours treats them as first-class citizens, i.e., accessible objects.
- This paves the way for higher-order functions.

Future Work

- Functions with predicate parameters.
- Additional means for defining functions such as currying.