Answer Sets Generated by Selector Functions

Sibylle Schwarz
Institut für Informatik, Universität Leipzig
Leipzig, Germany, Augustusplatz 10–11

Abstract

This paper presents a framework to define various declarative semantics of generalized logic programs (programs with nested expressions). Each of these semantics contains all models of a generalized logic program that can be reached from the least interpretation by a monotonic chain of interpretations. These chains are parameterized by selector functions. We identify selector functions that generate well-known classes of preferred models like minimal, stable, stable generated and least Herbrand models.
An interesting example of a selector-generated semantics is the end-regulated semantics. We study its connections to other semantics and the effect of some program transformations.

1 Motivation

Generalized logic programs are sets of rules that contain arbitrary quantifier free first order formulas in both their bodies and their heads. Definite, normal and disjunctive logic programs form proper subclasses of this program class.

The meaning of generalized logic programs is given by a declarative semantics. The program represents a set of logic formulas. One way to define a declarative semantics is to select a subset of the set of all models of these formulas.

The least Herbrand model of a definite program represents the intended meaning of a definite program in a unique and intuitive way. Things are not as easy if composed formulas are allowed in the program rules. Stable model semantics (Gelfond and Lifschitz, 1988) and, more generally, answer set semantics define the properties a model has to satisfy to be an intended model of a program.

In most approaches, answer sets are minimal models of a transformed program. Almost all generalizations of answer sets are based on stable models presented by Gelfond and Lifschitz in (Gelfond and Lifschitz, 1988). Essentially, for each extension to programs with more expressive syntax, an appropriate program transformation has to be found (Przymusinski, 1994), (Brass et al., 1998), (Sakama and Inoue, 1995). In this way, Lifschitz, Tang and Turner defined in (Lifschitz et al., 1999) stable models of generalized logic programs (programs with nested expressions).

This paper presents an alternative approach to extend the stable model semantics to generalized logic programs. This extension does not depend on program transformations. It is a generalization of stable generated models, defined by Herre and Wagner in (Herre and Wagner, 1997b). We select intended models by connecting them to zero-truth by a chain of interpretations.

In Section 2, some notations are recalled. The basic idea in our approach is the reachability of models by special chains of interpretations. These chains are defined in Section 3. In Section 4 we study some selector functions and their influence to models they generate. End-regulated models are an interesting example of selector generated models, that has been hardly examined so far. In Section 5 and Section 6 we study their properties and the effect of some program transformations.

2 Background

We assume that the reader is familiar with the standard notions of propositional and first order logic, such as atom, literal, formula, conjunctive and disjunctive normal form, Herbrand base B_P for a program P and the usual model relation $\models$ between formulas and two-valued interpretations. We use the connectives $\texttt{not}, \vee$ and $\wedge$.

Our framework defines the semantics of a logic program in terms of its ground instantiation. As usual in answer set semantics, we can restrict our attention to propositional programs. Atoms correspond to propositional variables and the base of a program is the set of all propositional variables that correspond to atoms occurring in the ground instantiation of the program.

Rules r in generalized logic programs are pairs of propositional formulas $\texttt{body}(r)$ and $\texttt{head}(r)$.
We denote rules by $\texttt{body}(r) \Rightarrow \texttt{head}(r)$. Note that the meaning of the rule arrow $\Rightarrow$ is different from classical implication. A *generalized logic program* is a set of rules.

Program classes with restricted rule syntax are

definite logic programs
$\texttt{body}(r) = \bigwedge_{i \in \{1,\dots,n\}} a_i$
all a_i and $\texttt{head}(r)$ are atoms

normal logic programs
$\texttt{body}(r) = \bigwedge_{i \in \{1,\dots,n\}} a_i \wedge \bigwedge_{j \in \{1,\dots,m\}} \texttt{not}\ b_j$
all a_i, b_j and $\texttt{head}(r)$ are atoms
We will use the notation
$\texttt{body}^+(r) := \{a_i \mid i \in \{1,\dots,n\}\}$ and
$\texttt{body}^-(r) := \{b_j \mid j \in \{1,\dots,m\}\}$

disjunctive logic programs
$\texttt{head}(r) = \bigvee_{i \in \{1,\dots,k\}} c_i \vee \bigvee_{j \in \{1,\dots,l\}} \texttt{not}\ d_j$
all c_i, d_j are atoms,
bodies as in normal logic programs

The sets $\texttt{head}(P)$ and $\texttt{body}(P)$ contain all heads and bodies of rules in P, respectively.

An *interpretation* $I : B_P \to W$ of a program is a total function from the base of the program to a set of truth values W.

Any ordering on the set of truth values induces an ordering on interpretations by pointwise comparison.

In this paper, two-valued interpretations ($W = \{\mathbf{t}, \mathbf{f}\}$ with the ordering $\mathbf{f} < \mathbf{t}$) are used. If convenient (mostly in examples), we identify an interpretation with its pre-image of $\mathbf{t}$. The set of all two-valued interpretations of the program P is denoted by $\mathbf{I}_P$.

Note that the framework is not restricted to this special case. The definitions given here are general enough to allow any complete lattice for W.

The interpretation $I_{\mathbf{f}}$ that maps each atom to $\mathbf{f}$ is the *least* interpretation. Given a set $\mathbf{I}$ of interpretations, the set of all its minimal elements is $\mathrm{Min}(\mathbf{I})$ and the least upper bound of $\mathbf{I}$ is denoted by $\sup \mathbf{I}$.

For *intervals* of interpretations we write

$$[I, J] := \{K \mid I \leq K \leq J\}$$

The set $\mathrm{Mod}(\Phi)$ of interpretations contains exactly all models of a set of formulas Φ. An interpretation I is a *model of a program rule* r (written $I \models r$), if it is a model of the formula $\texttt{not body}(r) \vee \texttt{head}(r)$ where $\texttt{not}$ is interpreted as classical negation. An interpretation is a *model of a logic program* P, if it is a model of all rules in P. A set of interpretations $\mathbf{I}$ satisfies a formula φ, if each interpretation in $\mathbf{I}$ satisfies φ (written $\mathbf{I} \models \varphi$).

A set $\mathbf{I}$ of interpretations selects the set of rules from a program P, whose bodies are satisfied in each interpretation from $\mathbf{I}$.

Definition 1

$$P_{\mathbf{I}} := \{r \in P \mid \mathbf{I} \models \texttt{body}(r)\}$$

The *immediate consequence operator* of a definite program P is $T_P : \mathbf{I}_P \to \mathbf{I}_P$ where
$T_P(I) = \{a \mid \exists r \in P : \texttt{head}(r) = a \text{ and } I \models \texttt{body}(r)\}$.

The *Gelfond-Lifschitz transformation* defined in (Gelfond and Lifschitz, 1988) depends on an interpretation M and maps a normal program P to its definite *reduct*

$$P^M := \{ \bigwedge \texttt{body}^+(r) \Rightarrow a \mid \\ \exists r \in P : \texttt{head}(r) = a \text{ and } \texttt{body}^-(r) \cap M = \emptyset\}$$

Stable models of a normal logic program P are the fixed points M of the operator T_{P^M} (Gelfond and Lifschitz, 1988).

3 Models Generated by Chains

In general, not every model of a logic program represents the intended meaning of this program. A subset of intuitively appropriate models has to be selected. These models should be

grounded (can be created bottom-up from zero truth by iterated application of program rules) and

supported (any atom in the model is justified by a program rule whose body is satisfied in this model).

In other generalizations of stable models to languages more expressive than normal programs, *minimality* is considered an important property of an intended model. We do *not* require minimality, because this not always seems to be intuitive.

Example 1. Someone is entering a cafeteria to order a glass of water or an espresso. Espresso comes with a glass of water. This situation can be represented by the program $P = \{\Rightarrow e \vee w, e \Rightarrow w\}$. P has the models $M_1 = \{w\}$ and $M_2 = \{e, w\}$. Both of them are intended, as the choice between espresso or water should not depend on the way they are served, and the decision for espresso does not mean a decision against water. But $M_1 = \{e, w\}$ is not a minimal model of P. In minimal model semantics it is impossible to have an espresso in this situation.

To guarantee groundedness of the selected models, we require that they can be reached from the least interpretation by iterative application of a successor function.

Depending on the program P, a guessed interpretation M, and a selector function $\mathtt{sel} : \mathbf{I}_P \rightarrow 2^{\mathbf{I}_P}$, we define chains of interpretations.

Definition 2 **($(\mathtt{sel}, P, M)$-chain)** *A* $(\mathtt{sel}, P, M)$-chain *is a sequence $(I_\alpha)_{\alpha \leq \kappa} \subseteq [I_\mathbf{f}, M]$ of Herbrand interpretations of the program P satisfying*

1. *$I_0 = I_\mathbf{f}$ and $I_\kappa = M$,*
2. *for successor ordinals α : $I_\alpha \in S^P_{\mathtt{sel}}(I_{\alpha-1})$,*
3. *for limit ordinals α : $I_\alpha = \sup\{I_\beta \mid \beta < \alpha\}$.*

Interpretations in a chain are connected by the successor function S, that works like a non-deterministic consequence operator.

Definition 3 **(Successor function $S^P_{\mathtt{sel}}$)**

$S^P_{\mathtt{sel}} : {\mathbf{I}_P}^2 \rightarrow 2^{\mathbf{I}_P}$

$S^P_{\mathtt{sel}}(M, I) := \mathrm{Min}\left(\mathrm{Mod}\left(\mathtt{head}\left(P_{\mathtt{sel}(M,I)}\right)\right) \cap [I, M]\right)$

By Definition 1 in the previous section, $P_{\mathtt{sel}(M,I)}$ is the set of all rules from P whose bodies are satisfied in each interpretaion in $\mathtt{sel}(M, I)$.

Remark. Because of the minimality of the successors we know that if an interpretation is a stable successor of itself ($I \in S^P_{\mathtt{sel}}(M, I)$) then it is the only successor ($S^P_{\mathtt{sel}}(M, I) = \{I\}$). Interpretations I with this property are called *fixed points* of $S^P_{\mathtt{sel}}(M, \bullet)$.

In the next section, we will study the influence of several selector functions on the sets of models they generate. Some prominent sets of intended models are special cases of the following definition.

Definition 4 **($\mathtt{sel}$-generated interpretation)** *We call an interpretation $M \in \mathbf{I}_P$ a* $\mathtt{sel}$-generated *interpretation of the program P if there is an $(S^P_{\mathtt{sel}}, M)$-chain and M is a fixed point of $S^P_{\mathtt{sel}}(M, \bullet)$.*

Note that a $\mathtt{sel}$-generated interpretation of P is not necessarily a model of P. To this end, $\mathtt{sel}$ has to satisfy an additional condition.

Lemma 1 *If $\mathtt{sel}(M, M) = \{M\}$ and M is a fixed point of $S^P_{\mathtt{sel}}(M, \bullet)$ then M is a model of P.*

Proof. $\{M\} = S^P_{\mathtt{sel}}(M, M)$. By Definition 3 this is $\{M\} = \mathrm{Min}\left(\mathrm{Mod}\left(\mathtt{head}\left(P_{\mathtt{sel}(M,M)}\right)\right) \cap [M, M]\right)$.
Hence $M \in \mathrm{Mod}\left(\mathtt{head}\left(P_{\mathtt{sel}(M,M)}\right)\right)$.
By assumption $\mathtt{sel}(M, M) = \{M\}$ this is
$M \in \mathrm{Mod}\left(\mathtt{head}\{r \in P \mid M \models \mathtt{body}(r)\}\right)$ and therefore $M \models P$. □

Definition 5 **($\mathtt{sel}$-generated semantics)** *The $\mathtt{sel}$-generated semantics of a generalized logic program is the set of all its $\mathtt{sel}$-generated models.*

4 Some Well-known Answer Sets and their Generating Selector Functions

In this section we analyze some selector functions. We will recognize some prominent model types that are generated by these functions.

The first candidate selects just the guessed interpretation.

Definition 6

$$\mathtt{sel}_\mathrm{u}(M, I) := \{M\}$$

The corresponding successor function applies the same set of rules in each step.

Lemma 2 *Any minimal model of a generalized program is $\mathtt{sel}_\mathrm{u}$-generated.*

Proof. We show, if M is a minimal model of P then it is $\mathtt{sel}_\mathrm{u}$-generated. M is a model of P iff
$M \in \mathrm{Mod}(\mathtt{head}(\{r \in P \mid M \models \mathtt{body}(r)\}))$ or, by definition of $P_{\mathtt{sel}_\mathrm{u}(M,M)}$, $M \in \mathrm{Mod}(\mathtt{head}(P_{\mathtt{sel}_\mathrm{u}(M,M)}))$.
Then $\{M\} = \mathrm{Min}\left(\mathrm{Mod}(\mathtt{head}(P_{\mathtt{sel}_\mathrm{u}}(M, M))) \cap [M, M]\right)$ and M is a fixed point of $S^P_{\mathtt{sel}}(M, \bullet)$.
As $\mathtt{sel}_\mathrm{u}$ ddoes not depend on the second argument $\mathtt{sel}_\mathrm{u}(M, M) = \mathtt{sel}_\mathrm{u}(M, I_\mathbf{f})$
As M is a minimal model of P, M is the only model of P in $[I_\mathbf{f}, M]$. Therefore
$\{M\} = \mathrm{Min}\left(\mathrm{Mod}(\mathtt{head}(P_{\mathtt{sel}_\mathrm{u}}(M, I_\mathbf{f}))) \cap [I_\mathbf{f}, M]\right)$. By Definition 3 this is $\{M\} = S^P_{\mathtt{sel}_\mathrm{u}}(M, I_\mathbf{f})$

Thus M is a $\texttt{sel}_\mathrm{u}$-successor of $I_\mathbf{f}$ and the chain $(I_\mathbf{f}, M)$ proves that M is $\texttt{sel}_\mathrm{u}$-generated. □

Note that not every $\texttt{sel}_\mathrm{u}$-generated model is a minimal model, as the following example shows.

Example 2. The model $M = \{a, c\}$ of program $P = \{\Rightarrow a \vee b, c \Rightarrow c\}$ is $\texttt{sel}_\mathrm{u}$-generated via the chain $(\emptyset, M)$. Both rules can fire in M, $P_{\texttt{sel}_\mathrm{u}}(M, \emptyset) = P_{\texttt{sel}_\mathrm{u}}(M, M) = P$ and M is a minimal extension of $\emptyset$ that satisfies both heads $a \vee b$ and c. However M is not a minimal model of P, as $\{a\} \subset M$ is a model of P.

Another function selects in each step the current interpretation.

Definition 7

$$\texttt{sel}_\mathrm{l}(M, I) := \{I\}$$

With this selector function the successor function works like the usual immediate consequence operator.

Lemma 3 *The only $\texttt{sel}_\mathrm{l}$-generated interpretation of a definite program P is the least Herbrand model of P.*

Proof. For all n, $T_P(I_n) = \{a \mid \exists r \in [P] : \texttt{head}(r) = a$ and $I_n \models \texttt{body}(r)\}$ is the unique extension that satisfies all rules whose bodies are true in I_n. Each $\texttt{sel}_\mathrm{l}$-chain starts in $I_0 = \emptyset$. Hence $T_P^n(\emptyset)$ and $(I_n)_{n \le \omega}$ are the same sequences and have the same fixed points. □

The next selector function allows to apply all rules that are satisfied in both the current interpretation I and the guessed interpretation M.

Definition 8

$$\texttt{sel}_\mathrm{lu}(M, I) := \{I, M\}$$

In some sense, this selector function mimics the Gelfond-Lifschitz transformation.

Lemma 4 *The set of all $\texttt{sel}_\mathrm{lu}$-generated interpretations of a normal logic program P contains exactly all stable models of P.*

Proof. Let M be $\texttt{sel}_\mathrm{lu}$-generated by the chain $(I_n)_{n \le \omega}$. We show that for all n, $I_{n+1} = T_{P^M}(I_n)$.
For $I \subseteq M$ it can be seen easily that $\{I, M\} \models \texttt{body}(r)$ iff $\texttt{body}^+(r) \subseteq I$ and $M \cap \texttt{body}^-(r) = \emptyset$.
The unique minimal extension of I_n is $I_{n+1} = \{a \mid \exists r \in [P] : \texttt{head}(r) = a, \texttt{body}^+(r) \subseteq I_n$ and $\texttt{body}^-(r) \cap M = \emptyset\}$. Each rule $\texttt{body}(r) \Rightarrow a$ in $[P]$, with $\texttt{body}^-(r) \cap M = \emptyset$ has a counterpart $\bigwedge \texttt{body}^+(r) \Rightarrow a$ in the reduct P^M.
Thus $I_{n+1} = \{a \mid \exists r \in P^M : \texttt{head}(r) = a, \texttt{body}^+(r) \subseteq I_n\} = T_{P^M}(I_n)$. Therefore T_{P^M} and $(I_n)_{n \le \omega}$ have the same fixed point M. □

The interval selector function requires the body of a firing rule to be satisfied in the whole interval between the current and the guessed interpretation.

Definition 9

$$\texttt{sel}_\mathrm{i}(M, I) := [I, M]$$

The definition of stable generated models by Herre and Wagner (Herre and Wagner, 1997b) is just another formulation of the condition to be a $\texttt{sel}_\mathrm{i}$-generated interpretation.

In (Herre and Wagner, 1997b) it is shown that for normal logic programs the sets of stable models and stable generated models coincide. Therefore $\texttt{sel}_\mathrm{i}$ also generates stable models of normal and least Herbrand models of definite programs.

It is a strong condition on a rule that its body must be satisfied in the whole interval from the current to the guessed interpretation.

Example 3. The model $M = \{a, b\}$ of the program $P = \{a \vee \texttt{not}\ b \Rightarrow a, a \Rightarrow b\}$ is not stable generated. The first rule cannot be applied in $\emptyset$, because its body is not satisfied in $\{b\} \in [\emptyset, M]$.

If a selector function $\texttt{sel}$ would allow the first rule to fire in $\emptyset$, there is the $(S^P_{\texttt{sel}}, M)$-chain $(\emptyset, \{a\}, \{a, b\})$. All its elements satisfy the body of the first rule.

This leads to the definition of the chain selector function. It has the complete chain for an additional parameter.

Definition 10

$$\texttt{sel}_\mathrm{c}(M, I, (I_n)_{n \le \omega}) := [I, M] \cap \{I_n \mid n \le \omega\}$$

Unlike in $\texttt{sel}_\mathrm{i}$, the body of a firing rule has to be persistent only with respect to the *chain* that generates the model.

Remark. Chains defined by one of the selector functions above reach their fixed points after at most ω steps. Hence the length of each chain can be restricted to ω without losing information. This is a straightforward generalization of Claim 1 in (Herre and Wagner, 1997b).

5 End-regulated Models

The selector function $\texttt{sel}_\mathrm{c}$ in Definition 10 generates an interesting class of models. In (Herre and Wagner, 1997a), $\texttt{sel}_\mathrm{c}$-generated models are called *end-regulated* but are not studied further.

Definition 11 *An* end-regulated model *of a generalized program P is a $\texttt{sel}_\mathrm{c}$-generated interpretation of P. The* end-regulated semantics *maps a generalized program to the set of all its end-regulated models.*

Like stable generated models, end-regulated models are a conservative extension of stable models.

Theorem 1 *The set of end-regulated models of a normal logic program P coincides with the set of all stable models of P.*

Proof. If a conjunction of literals is satisfied in both the upper and lower bound of an interval, then it is satisfied in each interpretation of this interval. As all bodies in a normal logic program are conjunctions of literals, in each step $P_{\texttt{sel}_\mathrm{lu}}$, $P_{\texttt{sel}_\mathrm{i}}$, $P_{\texttt{sel}_\mathrm{c}}$ (parameterized with the correct chain) contain exactly the same rules from a ground normal program P. Thus $\texttt{sel}_\mathrm{lu}$, $\texttt{sel}_\mathrm{i}$ and $\texttt{sel}_\mathrm{c}$ generate the same set of models.
From Lemma 4 we know that all stable models of normal programs are $\texttt{sel}_\mathrm{lu}$-generated, hence they are $\texttt{sel}_\mathrm{c}$-generated. □

The same argument proves that end-regulated and stable generated models coincide for any program that contains only rules whose bodies are conjunctions of literals. The heads may be arbitrarily complex. Disjunctive programs satisfy this syntactic restriction. Claim 10 in (Herre and Wagner, 1997b) says that stable models of disjunctive programs are stable generated. Both facts together prove the following theorem.

Theorem 2 *Each stable model of a disjunctive program P is an end-regulated model of P.* □

Example 1 (espresso or water) from Section 3 shows that the converse does not hold. The model $M_2 = \{e, w\}$ is an end-regulated model of the program $P = \{\Rightarrow e \vee w, e \Rightarrow w\}$ via the chain $(\emptyset, \{e\}, M)$, but not a minimal model of the reduct $P^M = P$.

There are interesting differences between stable generated and end-regulated models. A model can be end-regulated but not stable generated for two reasons.

The first case is illustrated by Example 3 from the previous section. The model $\{a, b\}$ is end-regulated via the chain $(\emptyset, \{a\}, \{a, b\})$, but not stable generated. The rule $a \vee \texttt{not}\ b \Rightarrow a$ cannot fire with respect to $\texttt{sel}_\mathrm{i}$ in $\emptyset$, because there is an interpretation $\{b\}$ *besides* the chain, that does not satisfy the body.

The second example utilizes the ability of chains to jump.

Example 4. $P = \{(\texttt{not}\ a \vee b) \wedge (a \vee \texttt{not}\ b) \Rightarrow a, \Rightarrow b\}$.
$M = \{a, b\}$ is end-regulated via the chain $(\emptyset, M)$ as the bodies of both rules are true in $\emptyset$ and in M.
The first rule cannot fire with respect to $\texttt{sel}_\mathrm{i}$ in $\emptyset$, because its body is false in $\{a\}$ and $\{b\}$.
Hence M is not stable generated.

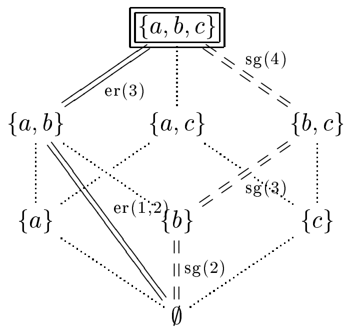
An interpretation J can be a $\texttt{sel}_\mathrm{c}$-successor of another interpretation I even if all interpretations in $[I, J]$ different from I and J do *not* satisfy the bodies of rules that are fired in I. As $\texttt{sel}_\mathrm{c}$ is only interested in the truth of the bodies on the chain, it is not affected by these interpretations. In this case, M is not stable generated, because interpretations *between* two links of the chain do not satisfy the body.

Even if a model of a generalized program is end-regulated and stable generated, the chains proving these properties need not be the same.

Example 5. The model $M = \{a, b, c\}$ is a stable generated and end-regulated model of

$$P = \{(\texttt{not}\ a \vee b) \wedge (a \vee \texttt{not}\ b) \Rightarrow a \quad (1), \\ \Rightarrow b \quad (2), b \Rightarrow c \quad (3), c \Rightarrow a \quad (4)\ \}.$$

But the chains that prove these properties differ.
The unique $S^P_{\texttt{sel}_\mathrm{i}}$-chain $(\emptyset, \{b\}, \{b, c\}, \{a, b, c\})$ proves that M is stable generated, and $(\emptyset, \{a, b\}, \{a, b, c\})$ is the unique $S^P_{\texttt{sel}_\mathrm{c}}$- chain proving M to be end-regulated. The following picture shows both chains in the interval $[\emptyset, M]$ with numbers of the applied rules.



With respect to the end-regulated semantics, rules of the form
$\varphi \vee \psi \rightarrow \psi$ can express that ψ is true in the intended model whenever φ becomes true at least once in the iterative construction of this model. The model itself need not satisfy φ. Such rules provide a nice way to express that once a formula ψ had been proved, it is kept in mind, but the proof itself is no longer interesting. Stable and stable generated model semantics do not provide this possibility as they do not always allow to apply the rule $\varphi \vee \psi \rightarrow \psi$ if φ is not true in the model itself.

6 Program Transformations

Two programs are said to be *end-regulated equivalent* (ER-equivalent) if they have the same set of end-regulated models.

The definition of end-regulated models does not depend on the inner structure of the bodies and heads of the program rules. Hence it is clear that any logically equivalent transformation of formulas can be applied separately to the body or the head of a rule. For instance, turning all bodies in disjunctive and all heads in conjunctive normal form does not change the end-regulated semantics of the program.

Splitting a rule with a conjunctive head leaves the end-regulated semantics unchanged.

Lemma 5 *The programs $P \cup \{\varphi \Rightarrow \psi_1 \wedge \psi_2\}$ and $P \cup \{\varphi \Rightarrow \psi_1, \varphi \Rightarrow \psi_2\}$ are end-regulated equivalent.*

Proof. We set $P_1 = P \cup \{\varphi \Rightarrow \psi_1 \wedge \psi_2\}$ and $P_2 = P \cup \{\varphi \Rightarrow \psi_1, \varphi \Rightarrow \psi_2\}$.
The interpretation M is an end-regulated model of P_1 if there is an $S^{P_1}_{\mathtt{sel}_c}$-chain $(I_n)_{n\leq\omega}$ for M. We show that this chain is also an $S^{P_2}_{\mathtt{sel}_c}$-chain for M.
If $I_n \models \varphi$, then I_{n+1} is a minimal extension of I_n that satisfies $\psi_1 \wedge \psi_2$ and a set F of heads from P. $F \cup \{\psi_1 \wedge \psi_2\}$ and $F \cup \{\psi_1, \psi_2\}$ have the same set of models and therefore the same set of minimal extensions of I_n. Hence each $(I_n)_{n\leq k}$ is an initial part of an $S^{P_2}_{\mathtt{sel}_c}$-chain for M. Thus $(I_n)_{n\leq\omega}$ is an $S^{P_2}_{\mathtt{sel}_c}$-chain for M.

The other direction can be proved analogously. □

Because any rule is equivalent to a rule with a head in conjunctive normal form, a generalized program can be transformed ER-equivalently to a program whose heads contain no conjunction.

Lifschitz, Tang and Turner proved in (Lifschitz et al., 1999) that the programs
$P \cup \{\varphi_1 \vee \varphi_2 \Rightarrow \psi\}$ and $P \cup \{\varphi_1 \Rightarrow \psi, \varphi_2 \Rightarrow \psi\}$ are equivalent with respect to their definition of stable models. Therefore in this sense every generalized logic program is equivalent to a disjunctive logic program.

Stable generated and end-regulated models are not invariant with respect to this transformation. This indicates that these semantics are more expressive than that from (Lifschitz et al., 1999).

Two examples illustrate the differences.

Example 6. Program $P = \{b \vee \mathtt{not}\ b \Rightarrow b\}$ has the stable generated and end-regulated model $\{b\}$. Program $Q = \{b \Rightarrow b, \mathtt{not}\ b \Rightarrow b\}$ has neither stable generated nor end-regulated models, because no body is satisfied in both interpretations $\emptyset$ and $\{b\}$.

In some cases, the end-regulated semantics is even less tolerant than the stable generated semantics. End-regulated models may differ on two programs even if the sets of stable generated models coincide.

Example 7. The interpretation $M = \{a, b\}$ is a model of both programs
$P = \{a \vee \mathtt{not}\ b \Rightarrow a, a \Rightarrow b\}$ and
$Q = \{a \Rightarrow a, \mathtt{not}\ b \Rightarrow a, a \Rightarrow b\}$.
Neither P nor Q have stable generated models, because no body is satisfied in all interpretations of $[\emptyset, \{a, b\}]$. But M is an end-regulated model of P via the chain $(\emptyset, \{a\}, \{a, b\})$ and M is not an end-regulated model of Q.

7 Further research

The presented framework is a starting point to many interesting investigations. The syntax of the rules in generalized programs can be extended to even more expressive rules. As the definitions of chain-generated models is very general, it generalizes to programs that contain different logical connectives. This can be done similarly to (Alferes et al., 2000) for the extensions of the stable generated model semantics to programs with two kinds of negation.

Similar to the lifting from propositional to first-order programs by grounding, selector-generated models can be defined for logic programs with constraints (Maher, 1999). This is a generalization of stable generated models to programs with constraints as done in (Schwarz, 2001).

As mentioned in the definition of interpretations, the definition of selector-generated models can be extended to many-valued models. It would be interesting to compare them to partial stable models (Przymusinski, 1991) and partial stable generated models (Alferes et al., 2000).

From (Przymusinski, 1990) we know that the well-founded semantics (Gelder et al., 1991) of a normal logic program can be defined by partial models. Analogously, it is possible to define a class of well-founded semantics based on partial selector-generated models of generalized programs.

Especially for end-regulated models, there is another interesting question. To prove a model to be end-regulated, we have to construct a chain. This construction already depends on the chain that is constructed, as the bodies of all firing rules have to be persistent on the upper part of this chain. So we have to guess the model (as in stable and stable generated models) and the chain. This is an additional source of non-determinism and increases the complexity of the test. It would be nice to have a bottom-up-algorithm to construct a chain that guarantees the persistence of the bodies during this construction.

8 Conclusions

We presented a framework to define several declarative semantics of generalized logic programs. These semantics are defined by selector functions that selects a set of interpretations from the interval from the least up to a guessed interpretation. Each of these semantics consists of the set of models that can be constructed by iterative application of the steps

1. Depending on the current and the guessed interpretation, the selector function chooses a set of interpretations.
2. Select all rules from program P, whose bodies are satisfied in all interpretations from 1.
3. Find all models of all heads of rules from 2, that are contained in the interval between the current and the guessed interpretation.
4. The successor is a minimal element of the set of interpretations from 3.

We found selector functions that generate minimal, stable, and stable generated models.

We studied properties of end-regulated models. They are a conservative extension of stable models of normal programs. It was shown that the end-regulated semantics is less tolerant with respect to program transformations than the stable generated model semantics. For programs containing rules with disjunctive bodies the end-regulated semantics is more fine-grained than Lifschitz, Tang and Turner's generalization of the stable semantics (Lifschitz et al., 1999).

References

Alferes, J. J., Herre, H., and Pereira, L. M. (2000). Partial models of extended generalized logic programs. In *Computational Logic*, pages 149–163.

Brass, S., Dix, J., Niemelä, I., and Przmusinski, T. C. (1998). A comparison of the static and the disjunctive well-founded semantics and its implementation. In Cohn, A. G., Schubert, L., and Shapiro, S. C., editors, *KR'98: Principles of Knowledge Representation and Reasoning*, pages 74–85. Morgan Kaufmann, San Francisco, California.

Gelder, A. V., Ross, K. A., and Schlipf, J. S. (1991). The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650.

Gelfond, M. and Lifschitz, V. (1988). The stable model semantics for logic programming. In Kowalski, R. A. and Bowen, K., editors, *Proceedings of the Fifth International Conference on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts. The MIT Press.

Herre, H. and Wagner, G. (1997a). Semantical foundation of extended generalized logic programs. Research Report 7, Universität Leipzig.

Herre, H. and Wagner, G. (1997b). Stable models are generated by a stable chain. *Journal of Logic Programming*, 30(2):165–177.

Lifschitz, V., Tang, L. R., and Turner, H. (1999). Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):369–389.

Maher, M. J. (1999). Adding constraints to logic-based formalisms. In Apt, K., Marek, V. W., Truszczyński, M., and Warren, D., editors, *The Logic Programming Paradigm: a 25 Years Perspective*, Artificial Intelligence Series, pages 313–331. Springer.

Przymusinski, T. (1990). Well-founded Semantics Coincides With Three-Valued Stable Semantics. *Fundamenta Informaticae*, XIII:445–463.

Przymusinski, T. (1991). Stable Semantics for Disjunctive Programs. *New Generation Computing Journal*, 9:401–424.

Przymusinski, T. (1994). Well-founded and stationary models of logic programs. *Annals of Mathematics and Artificial Intelligence*, 12:141–187.

Sakama, C. and Inoue, K. (1995). Paraconsistent stable semantics for extended disjunctive programs. *Journal of Logic and Computation*, 5(3):265–285.

Schwarz, S. (2001). Stable generated models of generalized constraint logic programs. In *Proceedings of the 10th International Workshop on Functional and (Constraint) Logic Programming (WFLP 2001)*. Universität Kiel.