Stable Generated Models of Generalized Constraint Logic Programs

Sibylle Schwarz

Institut für Informatik, Universität Leipzig,
Augustusplatz 10–11, 04109 Leipzig, Germany
schwarz@informatik.uni-leipzig.de

Abstract. We present a declarative semantics of generalized constraint logic programs based on stable generated models. Generalized logic programs contain arbitrary quantifier free formulas in the bodies and heads of their rules. Up to now the only declarative semantics for this program class is defined in terms of stable generated models introduced by Herre and Wagner.
A generalized constraint logic program additionally contains constraints in its rules. It combines the expressive power of constraint logic programs defined by Jaffar and Lassez and generalized logic programs. We define stable generated models of generalized constraint logic programs and describe some of their properties. This provides a declarative semantics of this program class that extends naturally the least $\mathcal{C}$–model semantics for definite programs with constraints defined by Jaffar et al.

1 Introduction

A declarative semantics provides a mathematically precise definition of the meaning of logic programs. We present a declarative semantics based on stable generated models for a very large class of logic programs with constraints.

Stable generated models. Stable generated models were introduced and studied by Herre and Wagener in [6]. They provide a declarative semantics for generalized logic programs with arbitrary quantifier free logic formulas in the bodies and heads of their rules. For this class of logic programs, this is the only declarative semantics defined so far.
Some well–known program classes are syntactically restricted special cases of the class of generalized logic programs. Thus a semantics for generalized logic programs applies to restricted classes, too.
Stable generated models of definite logic programs are just its least Hebrand models. For normal logic programs, there are different proposals of declarative semantics. The two–valued stable models and the three–valued well–founded models and their relationship are most important and well–understood for this program class. Although defined in a more general way, stable generated models are closely related to stable models defined by Gelfond and Lifschitz in [4]

for normal logic programs. Actually, stable generated models of a normal logic program are exactly the stable models of this program.
The main difference is that stable model semantics lays more stress on the groundedness of the intended models than on minimality, but stable models prefer minimality. This becomes evident in the disjunctive extension of stable models by Przymusinski [8], where every stable model is stable generated, but not every stable generated model is stable in the sense of [8]. There are disjunctive programs without stable models that have stable generated models.

Constraint logic programming. Constraint logic programs combine two declarative programming paradigms: logic programming and constraint solving. Classic logic programming fits in this framework with term equations as constraints and unification as solving algorithm.
Based on CLP as defined in [7], constraint extensions of logic programs can be defined in several ways. We can consider constraints as a new kind of atomic constructs and allow them in any place in a program rule, where an atom may occur. While this leads to a very expressive language for generalized programs, constraints are mostly used to formalize restrictions to the application of a rule as a whole. For this purpose, it is enough to add a constraint to the body of a rule. Because this simpler approach is considered to be appropriate, research goes in this direction and we adopt it here. Both extensions collapse to the CLP syntax for definite programs.
The declarative semantics of a definite constraint logic program is its least constraint model corresponding to the least Herbrand model of a definite program without constraints.
There have been several proposals for the extension of more expressive program classes by constraints. Some work has been done on well-founded models of constraint extensions of normal logic programs ([9]). Dix and Stolzenburg [2] give an operational semantics based on program transformations that approximates the well-founded semantics of disjunctive logic programs with constraints.

The paper has the following structure. We define some basic notations in section 2 and the syntax of several classes of logic programs with constraints in section 3. After introducing constraint interpretations, models and a special grounding function in section 4, we define stable generated models of programs with constraints in section 5 and give a detailed example in section 6. In section 7 we study some properties of stable generated models and compare them to a constraint extension of stable models for disjunctive programs in section 8.

2 Notations

A signature $\Sigma = (R, F)$ consists of a set R of relation symbols and a set F of function symbols. As constants can be considered as 0-ary functions, they are contained in F. $T_\Sigma(X)$ denotes the set of all terms built from Σ and variables from set X.

Atoms are the elements of the set $\mathsf{Atom}(\Sigma) = \{p(t_1, \ldots, t_n) \mid p \in R$ (n-ary) and $t_1, \ldots, t_n \in T_\Sigma(X)\}$.
For a set of logical connectives $J \subseteq \{\mathtt{not}, \wedge, \vee\}$, the logical language $L(\Sigma, J)$ is the least set containing all atoms in $\mathsf{Atom}(\Sigma)$ and being closed with respect to the following conditions:
If $\varphi \in L(\Sigma, J)$ and $\mathtt{not} \in J$ then $\mathtt{not}\ \varphi \in L(\Sigma, J)$.
If $\varphi, \psi \in L(\Sigma, J)$ and $* \in J \cap \{\wedge, \vee\}$ then $(\varphi * \psi) \in L(\Sigma, J)$.
Thus $\mathsf{Atom}(\Sigma) = L(\Sigma, \emptyset)$. Literals are elements of the set $L(\Sigma, \{\mathtt{not}\})$. $L^{\mathsf{Lit}}(\Sigma, J)$ denotes the set of formulas built from literals combined by connectives from J. Note that $L^{\mathsf{Lit}}(\Sigma, \{\wedge, \vee\}) \neq L(\Sigma, \{\mathtt{not}, \wedge, \vee\})$. As we will need it frequently, we write $\mathrm{Qf}(\Sigma) = L(\Sigma, \{\mathtt{not}, \wedge, \vee\})$ for the set of first order formulas without quantifiers.

3 Constraint logic programs

A *constraint system* $\mathcal{C}$ contains

- a logical language $L_\mathcal{C}$ on the constraint signature $\Sigma_\mathcal{C}$, whose elements are called *constraints*.
  $\Sigma_\mathcal{C}$ contains the binary relation symbol "=" and $\{\text{true}, \text{false}\} \subseteq L_\mathcal{C}$.
- a $\Sigma_\mathcal{C}$–structure $D_\mathcal{C} = (A, R_\mathcal{C}, F_\mathcal{C})$ called *constraint domain* satisfying the following properties:
  The carrier set A is not empty and we denote it by $|D_\mathcal{C}|$.
  $R_\mathcal{C}$ maps every n-ary relation symbol in $\Sigma_\mathcal{C}$ to a set of n-tuples from $|D_\mathcal{C}|$.
  $F_\mathcal{C}$ maps every n-ary function symbol in $\Sigma_\mathcal{C}$ to a function from $|D_\mathcal{C}|^n$ to $|D_\mathcal{C}|$.
  $D_\mathcal{C}$ satisfies the equality axioms for "=".

$\mathcal{C}$–program rules are built from two disjoint signatures, the constraint signature $\Sigma_\mathcal{C}$ and the program signature Σ_P containing just relation symbols. Thus $T_{\Sigma_P}(X) = X$ and $\mathsf{Atom}_P(\Sigma_P) = \{p(x_1, \ldots, x_n) \mid x_1, \ldots, x_n \in X$ and $p \in \Sigma_P$ (n-ary) $\}$. The corresponding sets of terms use the same infinite set X of Variables.
A $\mathcal{C}$–program rule r has three parts, the body $B(r)$ and the head $H(r)$, both contained in $\mathrm{Qf}(\Sigma_P)$, and the constraint $C(r) \in L_\mathcal{C}$. We denote a $\mathcal{C}$–program rule by $r = (C(r) \parallel B(r) \Rightarrow H(r))$.

Example 1. In this sense $r_1 = (x = 0 \wedge z = 3 + x \parallel p(x) \Rightarrow q(z))$ is a $\mathcal{C}$–program rule for $\{p, q\} \subseteq \Sigma_P$ and $\mathcal{C} = (\mathbb{N}, \{=, +\})$ while $r_2 = (x = 0 \parallel p(x) \Rightarrow q(3 + x))$ and $r_3 = (\text{true} \parallel p(0) \Rightarrow q(3))$ are not.

Later we will see that even terms from the constraint signature can be allowed as arguments to program relations in the body or head of a rule. Then we can handle rules like r_2 and r_3.
$\mathcal{C}$–program rules can be classified by restrictions on the set of formulas allowed in their body and head. Constraint logic programs are subsets of these classes of $\mathcal{C}$–program rules. Research on semantics of $\mathcal{C}$–programs has mostly been done on the classes

$\mathcal{C}$–LP : definite $\mathcal{C}$–programs, $H(r) \in \mathsf{Atom}(\Sigma_P), B(r) \in L(\Sigma_P, \{\wedge\})$ (in [7])
$\mathcal{C}$–NLP : normal $\mathcal{C}$–programs, $H(r) \in \mathsf{Atom}(\Sigma_P), B(r) \in L^{\mathsf{Lit}}(\Sigma_P, \{\wedge\})$
(for example in [3], [9])
$\mathcal{C}$–DLP : disjunctive $\mathcal{C}$–programs, $H(r) \in L(\Sigma_P, \{\vee\}), B(r) \in L^{\mathsf{Lit}}(\Sigma_P, \{\wedge\})$
(for example in [2])

In this paper we study the class $\mathcal{C}$–GLP of generalized $\mathcal{C}$–programs with $H(r)$ and $B(r) \in \mathrm{Qf}(\Sigma_P)$.

Example 2. $r = (x = z \wedge y = w \parallel p(x) \vee \texttt{not}\ p(y) \Rightarrow \texttt{not}\ (p(z) \wedge \texttt{not}\ q(w)))$ is a rule in $\mathcal{C}$–GLP. It is not contained in one of the restricted classes mentioned above.

4 $\mathcal{C}$–interpretations

Interpretation of constraints. To get a syntactic representation of domain elements, we have to expand the constraint signature by a constant for every element of $|D_\mathcal{C}|$. By slight misuse of notation we write $\Sigma_\mathcal{C}$ for the expanded signature, $L_\mathcal{C}$ for the expanded constraint language and d for the new constant representing element $d \in |D_\mathcal{C}|$ in the expanded signature.
Thus the constraint domain $D_\mathcal{C}$ given with $\mathcal{C}$ defines the constraint base

$$\begin{aligned}\mathrm{base}_\mathcal{C} = \{p(d_1, \ldots, d_n) \mid\ & p \text{ is a } n\text{-ary predicate symbol from } \Sigma_\mathcal{C} \\ & \text{and for } 1 \le i \le n : d_i \in |D_\mathcal{C}|\}\end{aligned}$$

The model relation $\models_\mathcal{C} \subseteq 2^{\mathrm{base}_\mathcal{C}} \times L_\mathcal{C}$ is given with the structure $D_\mathcal{C}$ by the mapping $R_\mathcal{C}$ of predicate symbols to tuples from $|D_\mathcal{C}|$. $I \models_\mathcal{C} p(d_1, \ldots, d_n)$ iff $(d_1, \ldots, d_n) \in R_\mathcal{C}(p)$.
An interpretation of a $\mathcal{C}$–program P with program signature $\Sigma_P = (R_P, \emptyset)$ expands $D_\mathcal{C}$ by a function that maps every n-ary relation symbol p from R_P to a set of n–tuples from $|D_\mathcal{C}|$.

$\mathcal{C}$–interpretations. We are only interested in interpretations I which are expansions of the constraint domain $D_\mathcal{C}$, that is $I \cap \mathrm{base}_\mathcal{C} = D_\mathcal{C}$. As in [7] the $\mathcal{C}$–base of P is the set

$$\begin{aligned}\mathrm{base}_{\mathcal{C},P} = \{p(d_1, \ldots, d_n) \mid\ & p \text{ is a } n\text{-ary predicate symbol from } \Sigma_P \\ & \text{and for } 1 \le i \le n : d_i \in |D_\mathcal{C}|\}\end{aligned}$$

and we call its elements *generalized atoms.*

Example 3. $P = \{r\}$ with r from Example 2 interpreted over domain $|D_\mathcal{C}| = \{0, 1\}$ defines $\mathrm{base}_{\mathcal{C},P} = \{p(0), p(1), q(0), q(1)\}$

$L_{\mathcal{C},P}$ denotes the language of ground formulas containing all generalized atoms of P and being closed with respect to $\{\texttt{not}, \vee, \wedge\}$.
A *$\mathcal{C}$–interpretation* is a set $I \subseteq \mathrm{base}_{\mathcal{C},P}$. The set of all $\mathcal{C}$–interpretations of program P on domain $D_\mathcal{C}$ is denoted by $\mathbf{I}_{\mathcal{C},P} = 2^{\mathrm{base}_{\mathcal{C},P}}$.
All $\mathcal{C}$–interpretations of a program P form a $\subseteq$–lattice.
An *interval* of $\mathcal{C}$–interpretations is denoted by $[I, J] = \{K \in \mathbf{I}_{\mathcal{C},P} \mid I \subseteq K \subseteq J\}$.

Grounding. Non–ground programs are related to structures via valuations $\beta : X \to |D_{\mathcal{C}}|$, which can be extended naturally to terms, formulas and $\mathcal{C}$–program rules. The set of all valuations is denoted by $B_{\mathcal{C}} = \{\beta : X \to |D_{\mathcal{C}}|\}$.
A constrained formula is a construct $(c \parallel \varphi)$ with $\varphi \in L(\Sigma_P, \{\mathtt{not}, \vee, \wedge\})$ and $c \in L_{\mathcal{C}}$. We define a grounding operator $[\ ]_{\mathcal{C}}$ that maps constrained formulas to sets of ground formulas respecting the constraint domain.
The set of generalized atoms $[(c \parallel a)]_{\mathcal{C}} = \{\beta(a) \mid \beta \in B_{\mathcal{C}}, D_{\mathcal{C}} \models_{\mathcal{C}} \beta(c)\}$ is the ground instantiation of a constrained atom $(c \parallel a)$ where $a \in \mathsf{Atom}_{\mathcal{C},P}$. Ground instances of a constrained formula are $[(c \parallel \varphi)]_{\mathcal{C}} = \{\beta(\varphi) \mid D_{\mathcal{C}} \models_{\mathcal{C}} \beta(c)\}$. For formulas φ without constraints we set $[\varphi]_{\mathcal{C}} = [(\text{true} \parallel \varphi)]_{\mathcal{C}} = \{\beta(\varphi) \mid \beta \in B_{\mathcal{C}}\}$.
Note that variables occurring in c but not in the formula φ are quantified existentially, as valuations are total functions on the infinite set of variables.

Example 4. For $|D_{\mathcal{C}}| = \{0, 1\}$ and $+$ and $=$ as usual on $\mathbb{Z}$
$[x = y \parallel p(x)]_{\mathcal{C}} = \{p(0),\ p(1)\}$ and $[y = x + 1 \parallel p(x)]_{\mathcal{C}} = \{p(0)\}$, but
$[y = x + 1 \wedge x = z + 1 \parallel p(x)]_{\mathcal{C}} = \emptyset$.

Ground instances of a constrained program rule are
$[(c \parallel \varphi \Rightarrow \psi)]_{\mathcal{C}} = \{\beta(\varphi) \Rightarrow \beta(\psi) \mid D_{\mathcal{C}} \models_{\mathcal{C}} \beta(c)\}$. The ground instantiation of a $\mathcal{C}$–program P is $[P]_{\mathcal{C}} = \bigcup_{r \in P} [r]_{\mathcal{C}}$.

Example 5. For P and $\mathcal{C}$ as in Example 3

$$
\begin{aligned}
[P]_{\mathcal{C}} = \{\ & p(0) \vee \mathtt{not}\ p(0) \Rightarrow \mathtt{not}\ \ (p(0) \wedge \mathtt{not}\ q(0))\,, \\
& p(0) \vee \mathtt{not}\ p(1) \Rightarrow \mathtt{not}\ \ (p(0) \wedge \mathtt{not}\ q(1))\,, \\
& p(1) \vee \mathtt{not}\ p(0) \Rightarrow \mathtt{not}\ \ (p(1) \wedge \mathtt{not}\ q(0))\,, \\
& p(1) \vee \mathtt{not}\ p(1) \Rightarrow \mathtt{not}\ \ (p(1) \wedge \mathtt{not}\ q(1))\}
\end{aligned}
$$

We can allow terms from the constraint signature as arguments to program relations in the body or head of a rule. A simple transformation can replace every constraint term t occurring in an argument of a program relation by a fresh variable $x \in X$ and add the constraint $x = t$ to $C(r)$. Because grounding is invariant under this transformation, the limitation to rules of the type defined above does not restrict expressiveness.

$\mathcal{C}$–models. The model relation $\models_{\mathcal{C},P} \subseteq \mathbf{I}_{\mathcal{C},P} \times L_{\mathcal{C},P}$ for $\mathcal{C}$–interpretations and ground formulas is defined by

$I \models_{\mathcal{C},P} a$	iff	$a \in I$	for $a \in \text{base}_{\mathcal{C},P}$
$I \models_{\mathcal{C},P} \mathtt{not}\ \varphi$	iff	$I \not\models_{\mathcal{C},P} \varphi$	
$I \models_{\mathcal{C},P} \varphi \wedge \psi$	iff	$I \models_{\mathcal{C},P} \varphi$ and $I \models_{\mathcal{C},P} \psi$	
$I \models_{\mathcal{C},P} \varphi \vee \psi$	iff	$I \models_{\mathcal{C},P} \varphi$ or $I \models_{\mathcal{C},P} \psi$	

A $\mathcal{C}$–interpretation $M \in \mathbf{I}_P^{\mathcal{C}}$ is a model of a ground $\mathcal{C}$– program rule $r = (B(r) \Rightarrow H(r))$ iff $M \models_{\mathcal{C},P}\ \mathtt{not}\ B(r) \vee H(r)$. M is a model for a non ground program rule r, if it is a model for all rules in $[r]_{\mathcal{C}}$. A $\mathcal{C}$–interpretation $M \in \mathbf{I}_P^{\mathcal{C}}$ is a model of $\mathcal{C}$-GLP–program P iff for all $r \in [P]_{\mathcal{C}} : M \models_{\mathcal{C},P} r$.

Example 6. $M_1 = \{q(0),\ q(1)\}$ and $M_2 = \{p(0),\ q(0),\ q(1)\}$ are models of P in Example 3, but $M_3 = \{p(0), p(1)\}$ is not.

We will make use of the observation $M \models_{\mathcal{C},P} P$ iff $M \models_{\mathcal{C},P} [P]_{\mathcal{C}}$.
For a set of $\mathcal{C}$–interpretations $\mathbf{I} \subseteq \mathbf{I}_{\mathcal{C},P}$ we write $\mathbf{I} \models_{\mathcal{C},P} \varphi$, iff for all $I \in \mathbf{I}$: $I \models_{\mathcal{C},P} \varphi$. This extends naturally to rules and programs.

5 Stable generated models of $\mathcal{C}$-GLP–programs

The instances in $[P]_{\mathcal{C}}$ do not contain $\mathcal{C}$–atoms and our grounding operator guarantees that all considered $\mathcal{C}$–interpretations respect the constraint domain $D_{\mathcal{C}}$. Some of the following definitions and lemmata do not depend on the constraint domain. We can generalize them quite naturally from the constraint–free case. Constraint bases may have a higher cardinality than the Herbrand base of a logic program. Fortunately all notions and results in [6] are general enough and apply to bases of higher cardinalities.
The definition of stable generated $\mathcal{C}$–models depends on minimal extensions of $\mathcal{C}$–interpretations. The next two lemmata will prove their existence in any step of our construction.

Lemma 1. *Let $\{I_\alpha \mid \alpha < \kappa\} \subseteq \mathbf{I}_{\mathcal{C},P}$, such that for all successor ordinals $\alpha < \kappa$: $I_{\alpha+1} \subseteq I_\alpha$ and for all limit ordinals $\lambda < \kappa : I_\lambda = \inf\{I_\beta \mid \beta < \lambda\}$, and a ground formula $\varphi = \bigvee_{a \in A} a \vee \bigvee_{b \in B}$ `not` b with finite sets of generalized atoms A and B. Then (for all $\alpha < \kappa : I_\alpha \models \varphi$) implies ($\bigcap_{\alpha<\kappa} I_\alpha \models \varphi$).*

Proof. For an indirect proof we assume :
for all $\alpha < \kappa : I_\alpha \models \varphi$) and $\bigcap_{\alpha<\kappa} I_\alpha \not\models \varphi$. This is only possible if
$B \subseteq \bigcap_{\alpha<\kappa} I_\alpha$ or $A \cap \bigcap_{\alpha<\kappa} I_\alpha = \emptyset$.

1. $B \subseteq \bigcap_{\alpha<\kappa} I_\alpha$ iff $\forall \alpha < \kappa : B \subseteq I_\alpha$ and then $\forall \alpha < \kappa : I_\alpha \not\models \bigvee_{b \in B}$ `not` b $\quad (*)$
2. $A \cap \bigcap_{\alpha<\kappa} I_\alpha = \emptyset$ iff $\forall a \in A\ \exists\ \alpha_a < \kappa : a \notin I_{n_a}$. As A is finite, $\{\alpha_a \mid a \in A\}$ is finite and there is an $\alpha < \kappa$ and for all $a \in A : \alpha_a < \alpha$. Then $I_\alpha \cap A = \emptyset$ and thus $I_\alpha \not\models \bigvee_{a \in A} a$.
   Together with $(*)$ this leads to $I_\alpha \not\models \bigvee_{a \in A} a \vee \bigvee_{b \in B}$ `not` b which contradicts the precondition $\forall \alpha < \kappa : I_\alpha \models \varphi$. □

Every formula $\varphi \in L_{\mathcal{C}}$ is logically equivalent to an conjunctive normal form $\varphi' = \bigwedge_{i=1}^{n} C_i$ and its models are the models of the finite set of formulas $\{C_i \mid 1 \leq i \leq n\}$. Hence Lemma 1 is even true for $\varphi \in L_{\mathcal{C}}$.

Lemma 2. *$\Phi \subseteq L_{\mathcal{C}}$ is a set of ground formulas, $I \subseteq \mathrm{base}_{\mathcal{C},P}$ a $\mathcal{C}$–interpretation and $M \supseteq I$ a $\mathcal{C}$–model of Φ.*
Then there exists a $\mathcal{C}$–model J satisfying the following conditions

1. *$J \in [I, M]$*
2. *for all $K \in [I, J] : K \models_{\mathcal{C},P} \Phi$ implies $K = J$ (minimality)*

Proof. The set of interpretations $P([I,M]) = [I,M] \cap \{K \mid K \models_{\mathcal{C},P} \Phi\}$ is not empty, as $M \in P([I,M])$.
Let $\{J_\alpha \mid \alpha < \kappa\} \subseteq P([I,M])$ be a decreasing sequence. By Lemma 1 its infimum $\bigcap_{\alpha<\kappa} J_\alpha$ is an element of $P([I,M])$.
Now by Zorn's lemma $P([I,M])$ contains minimal elements, satisfying conditions 1 and 2. $\square$

Interpretations are confirmed to be stable generated models by so called stable chains, where successive $\mathcal{C}$–interpretations are connected by a special successor relation.

Definition 1 (Successor relation $\mathrm{Succ}_{\mathcal{C}}^{(P,M)} \subseteq (\mathbf{I}_{\mathcal{C},P})^2$). *A $\mathcal{C}$–interpretation $J \in \mathbf{I}_{\mathcal{C},P}$ is a* stable successor *of a $\mathcal{C}$–interpretation $I \in \mathbf{I}_{\mathcal{C},P}$ with respect to a $\mathcal{C}$–interpretation $M \in \mathbf{I}_{\mathcal{C},P}$ and a program P (written $\mathrm{Succ}_{\mathcal{C}}^{(P,M)}(I,J)$), if the following properties hold*

(S1) $J \in [I,M]$
(S2) *for all $r \in [P]_{\mathcal{C}}$: if $[I,M] \models_{\mathcal{C},P} B(r)$ then $J \models_{\mathcal{C},P} H(r)$*
(S3) *(minimality) for all $K \subseteq J$: if K satisfies* **(S1)** *and* **(S2)** *then $K = J$*

$\mathrm{Succ}_{\mathcal{C}}^{(P,M)}$ has some useful properties:

- For any $J \subseteq M$ it defines a partial order on interval $[I,J]$.
- $\mathrm{Succ}_{\mathcal{C}}^{(P,M)}$ is monotonous.
- If $\mathrm{Succ}_{\mathcal{C}}^{(P,M)}(I,I)$ then there exists no $J \neq I : \mathrm{Succ}_{\mathcal{C}}^{(P,M)}(I,J)$.
  In this case we call I a fixed point of $\mathrm{Succ}_{\mathcal{C}}^{(P,M)}$.

An alternative definition without grounding would replace **(S2)** by
(S2') for all $r \in P$ and $\beta \in B_{\mathcal{C}}$, such that $D_{\mathcal{C}} \models_{\mathcal{C}} \beta(C(r)) : [I,M] \models_{\mathcal{C},P} \beta(B(r))$ implies $J \models_{\mathcal{C},P} \beta(H(r))$.
$\beta(H(r))$ is ground and w.l.o.g we can assume it to be a finite disjunction of ground literals. Then $J \models_{\mathcal{C},P} \beta(H(r))$ means $J \cap \beta(H(r)) \neq \emptyset$. Hence $\mathrm{Succ}_{\mathcal{C}}^{(P,M)}$ is a generalization of the immediate consequence operator $T_P^{\mathcal{C}}$ from [7].

Stable generated models are closely related to stable models for programs without constraints. They are defined similarly via a guess and check construction.

Definition 2 (Stable generated model). *$M \in \mathbf{I}_{\mathcal{C},P}$ is a* stable generated model *of a generalized constraint logic program P, if there is a set $\{I_\alpha \mid \alpha < \kappa\} \subseteq \mathbf{I}_{\mathcal{C},P}$ with the following properties*

(C1) $I_0 = \emptyset$
(C2) $M = \sup_{\alpha<\kappa} I_\alpha$
(C3) *for all successor ordinals $\beta < \kappa$:* $\mathrm{Succ}_{\mathcal{C}}^{(P,M)}(I_\beta, I_{\beta+1})$
(C4) *for all limit ordinals $\lambda < \kappa$:* $I_\lambda = \sup_{\beta<\lambda} I_\beta = \bigcup_{\beta<\lambda} I_\beta$

The we call M generated by the stable chain $(I_\alpha)_{\alpha<\kappa}$.

So we have to guess an interpretation M and if we can find a stable chain connecting the empty interpretation to M, the stability of M is proven.
Note that $\alpha < \beta < \kappa$ implies $I_\alpha \subseteq I_\beta$ and every stable chain is an increasing sequences of $\mathcal{C}$–interpretations.

6 Example

We use Example 4.9 from [2] to show how to prove an interpretation to be a stable generated model of a $\mathcal{C}$–program.

$$
\begin{array}{lrlll}
P = \{ & x = 0 & \| & \Rightarrow p(x) & (1) \\
 & y = x + 1 & \| \ \mathtt{not}\ p(x) \wedge \mathtt{not}\ p(y) & \Rightarrow p(x) & (2) \\
 & x > 0 \wedge y = x + 1 & \| & \Rightarrow p(x) \vee p(y) & (3) \\
 & y = 0 & \| \quad p(x) \wedge \mathtt{not}\ q(x) & \Rightarrow q(y) \} & (4)
\end{array}
$$

P is interpreted on $|D_{\mathcal{C}}| = \mathbb{N}$ with relations $\{=, >\}$ and function $+$ as usual.

We show that $M = \{q(0)\} \cup \{p(2n) \mid n \in \mathbb{N}\}$ is a stable generated model of P.

$$
\begin{array}{rllll}
[P]_{\mathcal{C}} = & \{ & \Rightarrow & p(0) & \} \ (1) \\
 & \cup \{ \ \mathtt{not}\ p(n) \wedge \mathtt{not}\ p(n+1) & \Rightarrow & p(n) & \mid n \in \mathbb{N} \} \ (2) \\
 & \cup \{ & \Rightarrow & p(n) \vee p(n+1) & \mid n > 0 \ \} \ (3) \\
 & \cup \{ \quad p(n) \wedge \mathtt{not}\ q(n) & \Rightarrow & q(0) & \mid n \in \mathbb{N} \} \ (4)
\end{array}
$$

$I_0 = \emptyset$
$[I_0, M]$ satisfies the bodies of all instances of rule 1 and rule 3. The body of rule 2 is not satisfied, as it is not satisfied in M. All stable successors of I_0 have to contain $p(0)$ (by rule 1) and $p(n)$ for one n of each pair of succeeding integers (by rule 3). The only way to satisfy the heads of all instantiations of rule 3 in $[I_0, M]$ is by
$I_1 = \{p(2n) \mid n \in \mathbb{N}\}$.
In $[I_1, M]$ rules 1, 3 and 4 are applicable (rule 4 only for instances by even numbers > 0). The heads of rules 1 and 3 are already satisfied and will be satisfied in every successor of I_1, so we have to extend I_1 by $\{q(0)\}$ and get
$I_2 = \{q(0)\} \cup \{p(2n) \mid n \in \mathbb{N}\}$, which is equal to M and a model of P. We have constructed a stable chain proving M to be stable generated.

Now we try to prove the stability of $M' = M \cup \{p(1)\}$. (Note that M' is a model of P.)
$I'_0 = \emptyset$. Same arguments as above, but now I'_1 has to be an element of $[I'_0, M']$. I_1 as above is the only *minimal* extension of I'_0 in $[I'_0, M']$. As I_1 satisfies all instantiated rules containing p–atoms in its head, no further extension of I_1 by p–atoms are possible in any successor of I_1.
Thus M' can not be reached by a stable chain and is not a stable generated model of P.

Remark 1. Let R be the set of all interpretations I that can be constructed in the following way:

1. w is an infinite word from the language L defined by the regular expression $1((1 \cup \epsilon)01)^\omega$ over signature $\Sigma = \{0, 1\}$. ϵ denotes the empty word and w_n the sign in the nth place in $w = w_0 w_1 \dots$.
2. $q(n) \in I$ iff $n = 0$ and $p(n) \in I$ iff $w_n = 1$.

We can construct M by the word $1(01)^\omega \in L$, thus $M \in R$. Every $I \in R$ is a stable generated model of P. The proofs are similar to that for M.
R' is the set of all interpretations I such that there exists a proper subset J of I in R. $M' \in R'$, because $M \in R$ and $M \subset M'$. All elements of R' are models of P, but none is stable generated by analogous proofs.
In fact R is exactly the set of all stable generated models of P.

7 Properties of stable generated models

Every stable generated $\mathcal{C}$–model is generated by a chain of length $\leq \omega$. As it does not depend on the constraint system, we can prove this fact analogously to [6].

Theorem 1. *If M is a stable generated $\mathcal{C}$–model of $P \subseteq \mathcal{C}$–GLP with stable chain $(I_\lambda)_{\lambda \leq \kappa}$ and $M = I_\kappa$, then there exists a $\beta \leq \omega$ such that $I_\beta = I_\kappa = M$.*

Recall that the definition of stable generated $\mathcal{C}$–models did not assume the guessed candidate interpretation M to be a model for P. Now we know that if M is confirmed to be stable generated by chain (I_n) then $M = I_\omega$ is a fixed point of the successor relation $\mathrm{Succ}_{\mathcal{C}}^{(P,M)}$ and satisfies **(S2)** from Definition 1. Hence for all $r \in [P]_{\mathcal{C}} : M \models_{\mathcal{C},P} B(r)$ implies $M \models_{\mathcal{C},P} H(r)$.
As every $\mathcal{C}$–interpretation respects $D_{\mathcal{C}}$ we have the following conclusion for non-ground programs.

Theorem 2. *Stable generated $\mathcal{C}$–models of a program $P \subseteq \mathcal{C}$–GLP are models of P.*

Stable generated $\mathcal{C}$–models are a natural generalization of least $\mathcal{C}$–models of $\mathcal{C}$–LP–programs defined in [7].

Theorem 3. *The only stable generated $\mathcal{C}$–model of a program $P \in \mathcal{C}$–LP is its least $\mathcal{C}$–model.*

Proof. M is a stable generated $\mathcal{C}$–model of $P \in \mathcal{C}$–LP generated by chain (I_n).
P is definite and all rules in P have atomic heads.
(S2') in Definition 1 is satisfied if for all $r \in P$ and for all valuations $\beta \in B_{\mathcal{C}}$ satisfying $D_{\mathcal{C}} \models_{\mathcal{C}} \beta(C(r))$ and $I_n \models_{\mathcal{C},P} \beta(B(r))$, the generalized atom $\beta(a)$ is contained in I_{n+1}.
$I_{n+1} = \{\beta(a) \mid (C(r) \parallel B(r) \Rightarrow a) \in P \text{ and } D_{\mathcal{C}} \models_{\mathcal{C}} \beta(C(r))\}$ is the unique minimal element satisfying **(S1)** and **(S2')**.
Hence $I_{n+1} = T_P^{\mathcal{C}}(I_n)$ defined in [7] and the elements I_n of the unique stable chain (I_n) correspond to the ordinal powers $T_P^{\mathcal{C}} \uparrow n$. Thus $I_\omega = T_P^{\mathcal{C}} \uparrow \omega$ and from Theorem 4.6 in [7] we know
$T_P^{\mathcal{C}} \uparrow \omega =$ least fixed point of $T_P^{\mathcal{C}} =$ least $\mathcal{C}$–model of P. ∎

8 Stable and stable generated models of positive disjunctive constraint logic programs

Stable models for normal logic programs were defined by Gelfond and Lifschitz in [4]. Like for stable generated models we have to guess an interpretation and prove it to be a stable model.

Definition 3 ([4]). *An interpretation M is a stable model of a normal logic program P, if it is the least Herbrand model of the program*

$$P^M = \left\{ \bigwedge_{b \in B^+} b \Rightarrow h \mid \bigwedge_{b \in B^+} b \wedge \bigwedge_{a \in B^-} \texttt{not } a \Rightarrow h \in P \text{ and } B^- \cap M = \emptyset \right\}$$

Przymusinski extended this notion to programs with a non–empty disjunction of atoms in the head of its rules (positive disjunctive programs) by the following definition.

Definition 4 ([8]). *An interpretation M is a stable models of a disjunctive logic program P, if it is one of the minimal Herbrand models of the transformed program P^M.*

We define stable models of positive disjunctive constraint logic programs and give an example for a stable generated model of a disjunctive constraint logic program P that is not a stable model of P.

Definition 5. *A $\mathcal{C}$–interpretation M is a stable model of a disjunctive constraint logic program P, if it is a minimal $\mathcal{C}$–model of*

$$P^M = \left\{ \bigwedge_{b \in B^+} b \Rightarrow \bigvee_{h \in H^+} h \mid \bigwedge_{b \in B^+} b \wedge \bigwedge_{b \in B^-} \texttt{not } b \Rightarrow \bigvee_{h \in H^+} h \in [P]_{\mathcal{C}} \text{ and } B^- \cap M = \emptyset \right\}$$

Example 7. $P = \{x \neq y \parallel \ \Rightarrow p(x) \vee q(y),\ x \neq y \parallel q(x) \Rightarrow p(y)\}$
Grounding P to $|D_{\mathcal{C}}| = \{0, 1\}$ gives
$[P]_{\mathcal{C}} = \{\Rightarrow p(0) \vee q(1),\ \Rightarrow p(1) \vee q(0),\ q(0) \Rightarrow p(1),\ q(1) \Rightarrow p(0)\}$
$M = \{p(0),\ p(1),\ q(0)\}$ is a stable generated model of P via the stable chain $I_0 = \emptyset, I_1 = \{p(0),\ q(0)\}, I_2 = \{p(0),\ p(1),\ q(0)\} = M$.
As P contains no negative body literals $P^M = P$. Because $\{p(0),\ p(1)\} \subset M$ is a model of P^M, M is not a minimal model of P^M and thus not a stable model of P.

But we can show analogously to [6]:

Theorem 4. *Every stable model of a positive disjunctive constraint logic pogram is stable generated.*

9 Further research

Generalized logic programs with constraints provide a powerful language for knowledge representation tasks. The proposed declarative semantics allows to describe mathematically exact the intended meaning of programs from this class. This paper marks a starting point for further investigations on semantics of this program class.

Three-valued stable generated models were defined in relation to extended logic programs with two kinds of negation (for instance in [1]). Transferring these definitions to generalized constraint programs, we can compare our partial stable generated model semantics to other three-value semantics for programs with constraints ([3]).

It was shown that the well-founded model of a normal logic program can be defined in terms of its partial stable generated models [5]. Using this result for normal programs with constraints we can compare our semantics to the well-founded model in [9] and even to CD–WFS by Dix and Stolzenburg [2] for disjunctive programs.

These comparisons will probably to develop an operational semantics corresponding to the presented declarative semantics. The development of a proof theoretical characterization of stable generated models and a query answering mechanism is surely the most important aim of future research in this area.

References

1. Jose Julio Alferes, Heinrich Herre, and Luis Moniz Pereira. Partial models of extended generalized logic programs. In *Computational Logic*, pages 149–163, 2000.
2. Jürgen Dix and Frieder Stolzenburg. A framework to incorporate non-monotonic reasoning into constraint logic programming. *Journal of Logic Programming*, 37(1-3):47–76, 1998. Special Issue on Constraint Logic Programming, Guest Editors: Kim Marriott and Peter Stuckey.
3. François Fages and Roberta Gori. A hierarchy of semantics for normal constraint logic programs. In Michael Hanus and Mario Rodríguez-Artalejo, editors, *Algebraic and Logic Programming, 5th International Conference, ALP'96*, volume 1139 of *lncs*, pages 77–91, Aachen, Germany, 25–27 September 1996. Springer.
4. Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth Bowen, editors, *Proceedings of the Fifth International Conference on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.
5. Heinrich Herre. Regular partial models and wellfounded semantics for generalized logic programs. *unpublished*.
6. Heinrich Herre and Gerd Wagner. Stable models are generated by a stable chain. *Journal of Logic Programming*, 30(2):165–177, February 1997.
7. Joxan Jaffar, Michael Maher, Kim Marriott, and Peter Stuckey. The sematics of constraint logic programs. *Journal of Logic Programming*, 37, 1998.
8. Teodor Przymusinski. Stable Semantics for Disjunctive Programs. *New Generation Computing Journal*, 9:401–424, 1991.
9. D. Toman. Computing the well-founded semantics for constraint extensions of Datalog. *Lecture Notes in Computer Science*, 1191:64–??, 1996.