A Uniform Treatment of Aggregates and Constraints in Hybrid ASP

Pedro Cabalar[1] and **Jorge Fandinno**[2] and **Torsten Schaub**[2] and **Philipp Wanko**[2]
[1] University of Corunna, Spain
[2] University of Potsdam, Germany

Abstract

Characterizing hybrid ASP solving in a generic way is difficult since one needs to abstract from specific theories. Inspired by lazy SMT solving, this is usually addressed by treating theory atoms as opaque. Unlike this, we propose a slightly more transparent approach that includes an abstract notion of a term. Rather than imposing a syntax on terms, we keep them abstract by stipulating only some basic properties. With this, we further develop a semantic framework for hybrid ASP solving and provide aggregate functions for theory variables that adhere to different semantic principles, show that they generalize existing aggregate semantics in ASP and how we can rely on off-the-shelf hybrid solvers for implementation.

Introduction

Many real-world applications have a heterogeneous nature that can only be captured by different types of constraints. This is commonly addressed by hybrid solving technology, most successfully in the area of Satisfiability modulo Theories (SMT; Nieuwenhuis, Oliveras, and Tinelli 2006). Meanwhile, neighboring areas like Answer Set Programming (ASP; Lifschitz 2008) follow suit. In doing so, they usually adopt the lazy approach to SMT that abstracts from specific constraints by interpreting them as opaque atoms. This integration is however often done in system-oriented ways that leave semantic aspects behind.

We first addressed this issue in (Cabalar et al. 2016) by providing a uniform semantic framework that allows us to capture the integration of ASP with foreign theories. This blends the non-monotonic aspects of ASP with other formalisms in a homogeneous representational framework. Moreover, it retains the representational aspects of ASP such as expressing defaults and an easy formulation of reachability, and transfers them to the integrated theory. In (Cabalar et al. 2020), we extended this to conditional aggregates, which already incurred a fraction of the aforementioned opaqueness principle. To illustrate this, consider the following hybrid ASP rule, taken from (Cabalar et al. 2020)[1]

$$\mathit{total}(R) := \mathit{sum}\dot{\{}\, \mathit{tax}(P) : \mathit{lives}(P,R)\, \dot{\}} \ \leftarrow\ \mathit{region}(R)$$

This rule gathers the total tax revenue of each region R by summing up the tax liabilities of the region's residents, P.

The need for subatomic structures emerges from the observation that the meaning of this rule should remain unchanged, in case the computation of the revenue is expressed using, for instance, a linear expression instead of the sum aggregate. However, this slight syntactic difference leads to a distinct constraint atom, whose semantics can be radically different. Only by inspecting the subatomic structure of both atoms, we can guarantee the expected behavior.

In this paper, we build an account of such abstract subatomic structures, namely *constraint terms*, and leverage them to provide a uniform treatment of linear constraints, conditional expressions, aggregates and similar future hybrid constructs. Furthermore, we investigate two different principles for conditional expressions: the *vicious circle principle* (vc) and a new one we call *definedness* (df). While vc has been investigated in traditional ASP in (Gelfond and Zhang 2019), this new principle ensures that the value of any conditional expression is always defined. This is different from vc according to which conditional expressions may be undefined due to cyclic dependencies (Cabalar et al. 2018). Interestingly, when combined with aggregates, the df principle leads to a generalization of another semantics known from ASP (Ferraris 2011), which provides the semantic underpinnings of aggregates used in the ASP system *clingo* (Gebser et al. 2019). Hence, for characterizing hybrid variants of *clingo*, this framework is a prime candidate. Moreover, we are able to show how, under certain circumstances, arithmetic aggregates (under both principles) can be mapped into conditional linear constraints under vc. Combined with our previous results (Cabalar et al. 2016; 2020), this allows us to use off-the-shelf constraint ASP (CASP; Lierler 2014) solvers to implement such hybrid extensions.

Here-and-There with Conditional Constraints

The syntax of the logic HT_C is based on a set of (constraint) variables $\mathcal{X}$ and constants or domain values from some non-

[1] We put dots on top of braces, viz. "$\dot{\{} \dots \dot{\}}$", to indicate *multisets*.

empty set $\mathcal{D}$. For convenience, we also distinguish a special symbol $\mathbf{u} \notin \mathcal{X} \cup \mathcal{D}$ that stands for *undefined*.

We introduce next what we call *basic* constraint terms, atoms and formulas and then extend these three concepts to incorporate conditional expressions. We define the set of *elementary terms* $\mathcal{T}^e \overset{\text{def}}{=} \mathcal{X} \cup \mathcal{D} \cup \{\mathbf{u}\}$, that is, variables, domain values and the symbol $\mathbf{u}$. Each theory will be defined over a given set of *basic (constraint) terms*, denoted as $\mathcal{T}^b$, that will include, at least, all elementary terms, i.e., $\mathcal{T}^e \subseteq \mathcal{T}^b$. The syntax of a basic term is left open, but can be any expression of infinite length. A *basic (constraint) atom* is an expression containing a (possibly infinite[2]) number of basic terms. For each theory, we assume a particular set of basic constraint atoms, denoted as $\mathcal{C}^b$. We do not impose any limitation on their syntax, though, in most cases, that syntax is defined by some grammar or regular pattern. For instance, *difference constraint atoms* are expressions of the form "$x - y \leq d$", containing the elementary terms x, y, d where $x, y \in \mathcal{X}$ are variables and $d \in \mathcal{D}$ a domain value. Note that we are free to define the subexpression "$x - y$" as a basic term or not, at our convenience. This does not affects the definition of "$x - y \leq d$" as a basic atom. The importance of distinguishing terms is thus not syntactic, but meta-logical: Distinguishing terms allows us to guarantee some properties that may not be satisfied on unstructured atoms.

A *basic formula* φ over $\mathcal{C}^b$ is defined as

$$\varphi ::= \bot \mid c \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \to \varphi \quad \text{where } c \in \mathcal{C}^b$$

We define $\top$ as $\bot \to \bot$ and $\neg\varphi$ as $\varphi \to \bot$ for every formula φ. We sometimes write $\varphi \leftarrow \psi$ instead of $\psi \to \varphi$ to follow logic programming conventions.

We now extend these notions to incorporate conditional constructs. A *conditional term* is an expression of the form

$$(s|s' \colon \varphi)$$

where s and s' are basic terms and φ is a basic formula. The intuitive reading of a conditional term is "get the value of s if φ holds, or the value of s' if it does not." Now, a *(constraint) term* is either a basic term, a conditional term or some (possibly infinite) expression involving basic and conditional terms. As before, a *(constraint) atom* is an expression involving a (possibly infinite) number of constraint terms. We denote the set of all constraint terms and atoms by $\mathcal{T}$ and $\mathcal{C}$, respectively.

A *formula* φ over $\mathcal{C}$ is defined as a basic formula above but with $c \in \mathcal{C}$ being an arbitrary constraint atom rather than a basic one. Given a constraint term, atom or formula α, we denote the set of variables occurring in α by $\mathit{vars}(\alpha) \subseteq \mathcal{X}$.

For the semantics, we start by defining the extended domain as $\mathcal{D}_{\mathbf{u}} \overset{\text{def}}{=} \mathcal{D} \cup \{\mathbf{u}\}$. A *valuation* v over $\mathcal{X}, \mathcal{D}$ is a function $v : \mathcal{X} \to \mathcal{D}_{\mathbf{u}}$ where $v(x) = \mathbf{u}$ represents that variable x is left undefined. Moreover, if $X \subseteq \mathcal{X}$ is a set of variables, valuation $v|_X : X \to \mathcal{D}_{\mathbf{u}}$ stands for the projection of v on X. A valuation v can be alternatively represented as the set $\{(x, v(x)) \mid x \in \mathcal{X}, v(x) \in \mathcal{D}\}$ by what no pair $(x, \mathbf{u})$ is in the set. This representation allows us to use standard set inclusion for comparison. We thus write $v \subseteq v'$ to mean that

$$\{(x, v(x)) \mid x \in \mathcal{X}, v(x) \in \mathcal{D}\} \subseteq \{(x, v'(x)) \mid x \in \mathcal{X}, v'(x) \in \mathcal{D}\}$$

The set of all valuations over $\mathcal{X}, \mathcal{D}$ is denoted by $\mathcal{V}_{\mathcal{X},\mathcal{D}}$ and $\mathcal{X}, \mathcal{D}$ dropped whenever clear from context.

We define the semantics of basic constraint atoms via *denotations*, which are functions $[\![\cdot]\!] : \mathcal{C}^b \to 2^{\mathcal{V}}$, mapping each basic constraint atom to a set of valuations. For instance, each difference constraint like $x - y \leq d$ can be captured by a constraint atom "$x - y \leq d$" whose denotation $[\![\text{"}x - y \leq d\text{"}]\!]$ is given by the expected set:

$$\{v \in \mathcal{V} \mid v(x), v(y), d \in \mathbb{Z},\ v(x) - v(y) \leq d\} \qquad (1)$$

Satisfaction of constraint atoms involving conditional terms is defined by a previous syntactic *unfolding* of their conditional terms, using some interpretation to decide the truth values of formulas in conditions. Formally, an *interpretation* over $\mathcal{X}, \mathcal{D}$ is a pair $\langle h, t \rangle$ of valuations over $\mathcal{X}, \mathcal{D}$ such that $h \subseteq t$. The interpretation is *total* if $h = t$. With this, we define next two valuation functions for conditional terms, one following the *vicious cycle principle* (vc) and another ensuring the *definedness* of conditional terms (df).

Definition 1 *Given an interpretation $\langle h, t \rangle$ and a conditional term $s = (s'|s'' \colon \varphi)$, we define:*

$$vc_{\langle h,t \rangle}(s) = \begin{cases} s' & \text{if } \langle h, t \rangle \models \varphi \\ s'' & \text{if } \langle t, t \rangle \not\models \varphi \\ \mathbf{u} & \text{otherwise} \end{cases} \qquad (2)$$

$$\mathit{df}_{\langle h,t \rangle}(s) = \begin{cases} s' & \text{if } \langle h, t \rangle \models \varphi \\ s'' & \text{if otherwise} \end{cases} \qquad (3)$$

Note that the valuation functions rely on the satisfaction relation $\models$ defined below.

To illustrate the different behavior of vc and df, take the following simple example

$$x = 1 \ \leftarrow\ (1|0 \colon x = 1) \geq 0 \qquad (4)$$

stating that x must have value 1 when the conditional expression $(1|0 \colon x = 1) \geq 0$ holds. We see below that this rule has no stable model under vc-semantics while it has a unique one with $t(x) = 1$ under df-semantics. We face here a completely analogous situation to the following standard (non-hybrid) ASP rule with an aggregate:

$$\mathit{holds}_x(1) \ \leftarrow\ \mathit{count}\dot{\{}1 : \mathit{holds}_x(1)\dot{\}} \geq 0 \qquad (5)$$

where predicate atom $\mathit{holds}_x(1)$ is playing the role of $x = 1$ in (4). Rule (5) has the unique stable model $\{\mathit{holds}_x(1)\}$ under Ferraris' semantics for aggregates, which does not comply with vc, whereas it has no stable model[3] under Gelfond and Zhang's semantics, which satisfies vc.

Observation 1 *Given some total interpretation $\langle t, t \rangle$ and any conditional term s, we have $vc_{\langle t,t \rangle}(s) = \mathit{df}_{\langle t,t \rangle}(s)$.*

[2]An atom may have an infinite number of terms of infinite length. This cannot be represented as a string, but is still a expression similar, for instance, to some formula in infinitary logics.

[3] Gelfond and Zhang's semantics is defined exclusively for set based aggregates, but lifting it to multi-sets is straightforward.

Hence, for total interpretations, we may just write $eval_t(s)$ instead of $vc_{\langle t,t\rangle}(s)$ and $df_{\langle t,t\rangle}(s)$. In our running example, if $\langle t,t\rangle$ is a total interpretation such that $\langle t,t\rangle \models (x=1)$, then $eval_t(1|0\colon x=1)$ is 1. This means that to evaluate whether $\langle t,t\rangle$ satisfies (4) wrt any of the two semantics, we replace the conditional expression by the domain element 1 and, thus, evaluate whether $\langle t,t\rangle$ satisfies the basic formula

$$x = 1 \leftarrow 1 \geq 0 \tag{6}$$

which obviously holds since we assumed $\langle t,t\rangle \models (x=1)$. For non-total interpretations, the valuation is slightly more involved, so we will resume our example after introducing the definition of the satisfaction relation.

We permit that different occurrences of conditional expressions are interpreted according to different valuation functions (vc or df). This can be simply achieved by some syntactic distinction like, for instance, enclosing the expression with $(\cdot)$ for vc and with $[\cdot]$ for df. This allows us assigning different interpretations to different occurrences of the same expression, e.g., in the formula

$$x = 1 \leftarrow (1|0\colon x=1) \geq 0 \vee \neg[1|0\colon x=1] \geq 0 \tag{7}$$

In order to abstract from the particular syntax used, we just assume that there exists some selection function κ that tells us, for each occurrence of a conditional term s, which evaluation function must be used, that is, either $\kappa_{\langle h,t\rangle}(s) = vc_{\langle h,t\rangle}(s)$ or $\kappa_{\langle h,t\rangle}(s) = df_{\langle h,t\rangle}(s)$.

For a constraint atom $c \in \mathcal{C}$, we define $\kappa_{\langle h,t\rangle}(c)$ as the basic constraint atom that results from replacing each conditional term s in c by the basic term $\kappa_{\langle h,t\rangle}(s)$.

Definition 2 *Given a denotation $[\![\cdot]\!]$, a selection function κ, an interpretation $\langle h,t\rangle$* satisfies *a formula φ, written $\langle h,t\rangle \models_\kappa \varphi$, if*

1. $\langle h,t\rangle \models_\kappa c$ if $w \in [\![\kappa_{\langle w,t\rangle}(c)]\!]$ for $w \in \{h,t\}$

2. $\langle h,t\rangle \models_\kappa \varphi \wedge \psi$ if $\langle h,t\rangle \models_\kappa \varphi$ and $\langle h,t\rangle \models_\kappa \psi$

3. $\langle h,t\rangle \models_\kappa \varphi \vee \psi$ if $\langle h,t\rangle \models_\kappa \varphi$ or $\langle h,t\rangle \models_\kappa \psi$

4. $\langle h,t\rangle \models_\kappa \varphi \to \psi$ if $\langle w,t\rangle \not\models_\kappa \varphi$ or $\langle w,t\rangle \models_\kappa \psi$ for $w \in \{h,t\}$

We say that $\langle h,t\rangle$ is a κ-model of φ when $\langle h,t\rangle \models_\kappa \varphi$. In particular, vc- and df-models are those corresponding to evaluating all conditional terms according to vc or df, respectively. Furthermore, we may just write $\langle h,t\rangle \models \varphi$ when φ is a basic formula or when $\langle h,t\rangle$ is total, because the valuation function becomes irrelevant in those cases. Note that this satisfaction relation without subindex is the one used in Definition 1 for the valuation function. In the rest of the paper, we assume a fixed underlying denotation for constraint atoms. If not explicitly stated otherwise, we also assume a fixed underlying selection function.

It is worth noting that Definition 2 differs from (Cabalar et al. 2020) in Condition 1, which in our setting corresponds to:

1'. $\langle h,t\rangle \models_{vc} c$ if $h \in [\![vc_{\langle h,t\rangle}(c)]\!]$

That is, satisfaction of an atom was only checked on the here world h and the selection function was fixed to vc. In fact, the satisfaction relation was not parameterized with κ, since the unique valuation function used was vc. The following result[4] states that our semantics parameterized with the vc mapping actually corresponds to the semantics we introduced in (Cabalar et al. 2020).

Proposition 1 *Let φ be a formula and $\langle h,t\rangle$ be some interpretation. Then, $\langle h,t\rangle \models_{vc} \varphi$ iff $\langle h,t\rangle$ is a model of φ according to (Cabalar et al. 2020).*

To illustrate satisfaction of formulas under vc, take again (4) and suppose we have some $\langle h,t\rangle$ where $h(x) = \mathbf{u}$ and $t(x) = 1$. Then, $\langle t,t\rangle$ satisfies $x = 1$ and, as we saw above, this implies that $\langle t,t\rangle$ satisfies (4). On the other hand, we also can see that $\langle h,t\rangle$ does not satisfy $x = 1$ and, by definition, we get: $\langle h,t\rangle \models_{vc} (1|0\colon x=1) \geq 0$ iff both $\langle h,t\rangle \models \mathbf{u} \geq 0$ and $\langle t,t\rangle \models 1 \geq 0$. In fact, in view of Proposition 1, it is enough to check whether $\langle h,t\rangle$ satisfies $\mathbf{u} \geq 0$. That is, $\langle h,t\rangle \models (4)$ iff $\langle h,t\rangle$ satisfies the formula

$$x = 1 \leftarrow (\mathbf{u} \geq 0) \tag{8}$$

which holds because $\langle h,t\rangle \not\models (\mathbf{u} \geq 0)$.

A *theory* is a set of formulas. An interpretation $\langle h,t\rangle$ is a *κ-model* of some theory Γ, written $\langle h,t\rangle \models_\kappa \Gamma$, when $\langle h,t\rangle \models_\kappa \varphi$ for every $\varphi \in \Gamma$. A formula φ is a *tautology* (wrt some underlying denotation and selection function) when $\langle h,t\rangle \models_\kappa \varphi$ for every interpretation $\langle h,t\rangle$. Note that, this implies that a basic constraint atom $c \in \mathcal{C}^b$ is tautologous whenever $[\![c]\!] = \mathcal{V}$.

Definition 3 *A (total) interpretation $\langle t,t\rangle$ is a κ-*equilibrium model *of a theory Γ, if $\langle t,t\rangle \models_\kappa \Gamma$ and there is no $h \subset t$ such that $\langle h,t\rangle \models_\kappa \Gamma$.*

Valuation t is also called a *κ-stable model* of a set of formulas Γ when $\langle t,t\rangle$ is an κ-equilibrium model of Γ. For the case of vc-stable we get the next result.

Corollary 1 *The vc-stable models of any theory coincide with its stable models according to (Cabalar et al. 2020).*

Following with our example above, it is easy to see that the interpretation we had, $\langle h,t\rangle$ with $t(x) = 1$ and $h(x) = \mathbf{u}$, is not a vc-stable model of (4) because $\langle h,t\rangle \models_{vc} (4)$. If we consider, instead, the df-semantics, we will see that t is in fact a df-stable model of (4). This is because no $h' \subset t$ forms a model $\langle h',t\rangle$. We will prove it for $h' = h$ where $h(x) = \mathbf{u}$ and the proof for other interpretations is similar. First, note that for the df-semantics, Condition 1 of Definition 2 always uses the evaluation in both worlds h and t. It does not suffice with using h, as happened with vc (Proposition 1). Hence, to satisfy $\langle h,t\rangle \models_{df} (1|0\colon x=1) \geq 0$ we need both $\langle h,t\rangle \models 0 \geq 0$ and $\langle t,t\rangle \models 1 \geq 0$. As a result, $\langle h,t\rangle \models_{df} (4)$ iff $\langle h,t\rangle$ satisfies the formula

$$x = 1 \leftarrow (0 \geq 0) \wedge \neg\neg(1 \geq 0) \tag{9}$$

which does not hold. Just note that the right hand side is a tautology and that $\langle h,t\rangle$ does not satisfy $x = 1$ because $h(x) = \mathbf{u}$. Hence, t is a df-stable model of (4). The following result generalizes this double negation formalization.

Proposition 2 *Let $\langle h,t\rangle$ be an interpretation and $c \in \mathcal{C}$ be a constraint atom.*

Then, $\langle h,t\rangle \models_\kappa c$ iff $\langle h,t\rangle \models \kappa_{\langle h,t\rangle}(c) \wedge \neg\neg eval_t(c)$.

[4]An extended version of the paper including all proofs can be found here: https://arxiv.org/abs/2003.04176

The reason why we need to check both worlds for the df-semantics is to keep the *persistence property* of HT as stated in the following result.

Proposition 3 (Persistence) *Let $\langle h, t\rangle$ and $\langle t, t\rangle$ be two interpretations, and φ be a formula.*
Then, $\langle h, t\rangle \models_\kappa \varphi$ implies $\langle t, t\rangle \models_\kappa \varphi$.

The need for the additional evaluation in t comes from the fact that, under df valuation, some constraint atoms c may satisfy $h \in [\![\, \mathit{df}_{\langle h,t\rangle}(c)\,]\!]$ but $t \notin [\![\, \mathit{df}_{\langle h,t\rangle}(c)\,]\!]$, and so, if we only used h, persistence would be violated. To illustrate this feature, take the conditional constraint atom

$$(1|2\colon x = 1) \;\geq\; 2 \tag{10}$$

and, again, interpretation $\langle h, t\rangle$ with $h(x) = \mathbf{u}$ and $t(x) = 1$. Then, we obtain

$$\begin{aligned} \mathit{df}_{\langle h,t\rangle}(1|2\colon x = 1) &= 2 \\ \mathit{eval}_t(1|2\colon x = 1) &= 1 \end{aligned}$$

because $\langle h, t\rangle \not\models (x = 1)$ and $\langle t, t\rangle \models (x = 1)$. But then

$$\begin{aligned} h &\in [\![\, \mathit{df}_{\langle h,t\rangle}(1|2\colon x = 1) \geq 2\,]\!] \\ t &\notin [\![\, \mathit{eval}_t(1|2\colon x = 1) \geq 2\,]\!] \end{aligned}$$

The following proposition tells us that some other usual properties of HT are still valid in this new extension. Let us introduce some notation first. Given any HT formula φ, let $\varphi[\overline{a}/\overline{\alpha}]$ denote the uniform replacement of atoms $\overline{a} = (a_1, \dots, a_n)$ in φ by HT_C formulas $\overline{\alpha} = (\alpha_1, \dots, \alpha_n)$.

Proposition 4 *Let $\langle h, t\rangle$ and $\langle t, t\rangle$ be two interpretations, and φ be a formula. Then,*

1. *$\langle h, t\rangle \models_\kappa \varphi \to \bot$ iff $\langle t, t\rangle \not\models_\kappa \varphi$,*
2. *If φ is an HT tautology then $\varphi[\overline{a}/\overline{\alpha}]$ is an HT_C tautology.*

As an example of Property 2 in Proposition 4, we can conclude, for instance, that

$$(x - (y|3\colon p) \leq 4) \to \neg\neg(x - (y|3\colon p) \leq 4)$$

is an HT_C tautology because we can replace a in the HT tautology $a \to \neg\neg a$ by the HT_C formula $(x - (y|3\colon p) \leq 4)$. In particular, the second statement guarantees that all equivalent rewritings in HT are also applicable to HT_C.

Terms and assignments

As said before, the use of terms as subexpressions will be convenient to derive structural properties of constraint atoms. We will sometimes refer to a constraint atom using the notation $c[s]$ meaning that the expression for c contains some distinguished occurrence of subexpression s. We further write $c[s/s']$ to represent the syntactic replacement in c of subexpression s by s'. Then, we assume the following syntactic properties:

1. if $s \in \mathcal{T}$ is a term, then there are constraint atoms in $\mathcal{C}$ of the form $s = s$ and $s = d$ for every domain element $d \in \mathcal{D}$,
2. if $s \in \mathcal{T}$ is a term, $c[s] \in \mathcal{C}$ is a constraint atom and $s' \in \mathcal{T}^e$, then $c[s/s'] \in \mathcal{C}$ is also a constraint atom,
3. if $s, s' \in \mathcal{T}$ are terms such that s' is a subexpression of s and $c[s] \in \mathcal{C}$ is a constraint atom, then $c[s/s'] \in \mathcal{C}$.

Intuitively, Condition 1 states that we always define, at least, equality constraint atoms that allow comparing a term s to any domain element or to itself.

Atom $s = s$ is not a tautology: it is satisfied iff s has some value. For this reason, we will sometimes abbreviate $s = s$ as $\mathit{def}(s)$ meaning that s is defined. Conditions 2 and 3 state that replacement of terms must lead to syntactically valid expressions. Contrarily to first order logic, we only require that terms can be replaced by some particular class of terms rather than all possible terms. In particular, Condition 2 implies that replacing any term by an elementary term must lead to syntactically valid expressions. Condition 3 is similar but for every term that is also a subexpression. For instance, if $x - y$ and $x - y + z$ are terms and $x - y + z \leq 4$ is a constraint atom, then we must allow for forming constraint atoms $x \leq 4$ and $x - y \leq 4$ and $\mathbf{u} + z \leq 4$ among others. Note that, since $x - y + z$ is not a subexpression of $x - y$, we do not require $x - y + z + z \leq 4$ to be a constraint atom.

These intuitions are further formalized by imposing the following semantic properties for any denotation $[\![\cdot]\!]$, basic atom $c \in \mathcal{C}^b$, basic term $s \in \mathcal{T}^b$, domain element $d \in \mathcal{D}$, variable $x \in \mathcal{X}$, and any pair of valuations $v, v' \in \mathcal{V}$:

5. if $v(x) = v'(x)$ for all $x \in \mathit{vars}(c)$ then $v \in [\![\, c\,]\!]$ iff $v' \in [\![\, c\,]\!]$.
6. $v \in [\![\, c\,]\!]$ and $v \subseteq v'$ imply $v' \in [\![\, c\,]\!]$,
7. $v \in [\![\, c[s/\mathbf{u}]\,]\!]$ implies $v \in [\![\, c[s]\,]\!]$
8. $[\![\, d = d\,]\!] = \mathcal{V}$,
9. $[\![\, x = d\,]\!] = \{v \in \mathcal{V} \mid v(x) = d\}$,
10. $[\![\, s = d\,]\!] \cap [\![\, s = d'\,]\!] = \emptyset$ for any $d' \in \mathcal{D}$ with $d \neq d'$
11. if $v \in [\![\, s = s'\,]\!]$ for any term $s' \in \mathcal{T}$, then there is some $d' \in \mathcal{D}$ such that $v \in [\![\, s = d'\,]\!]$ and $v \in [\![\, s' = d'\,]\!]$
12. if $v \in [\![\, s = d\,]\!]$, then $v \in [\![\, c[s]\,]\!]$ iff $v \in [\![\, c[s/d]\,]\!]$,
13. if $v \notin [\![\, s = s\,]\!]$ and $v \in [\![\, c[s]\,]\!]$, then $v \in [\![\, c[s/\mathbf{u}]\,]\!]$.

Condition 5 asserts that the denotation of c is fixed by combinations of values for $\mathit{vars}(c)$, while other variables may vary freely, consequently becoming irrelevant. Condition 6 makes constraint atoms behave monotonically. Condition 7 is the counterpart of Condition 6 for terms. Intuitively, it says that, if a constraint does not hold for some term, then it cannot hold when that term is left undefined. For instance, if we include a constraint atom $x - (y|z\colon p) \leq 4$, then we must allow for forming the three constraint atoms $x - y \leq 4$ and $x - z \leq 4$ and $x - \mathbf{u} \leq 4$, too, and any valuation for the latter must also be a valuation for the former two. Conditions 8-13 describe the behavior of equality atoms. Conditions 12 and 13 respectively tell us that, given some valuation v, a term s can always be replaced by its defined value $v(s) = d$ or by $\mathbf{u}$, if it has no value. Reflexivity and symmetry of '$=$' can be derived from Conditions 11-12, as stated below.

Proposition 5 *Given terms s, s' and s'', the following conditions hold:*

1. *if $v \in [\![\, s = s'\,]\!]$ and $v \in [\![\, s' = s''\,]\!]$, then $v \in [\![\, s = s''\,]\!]$.*
2. *if $v \in [\![\, s = s'\,]\!]$, then $v \in [\![\, s' = s\,]\!]$, and*
3. *$[\![\, \mathbf{u} = s\,]\!] = [\![\, s = \mathbf{u}\,]\!] = [\![\, \mathbf{u} = \mathbf{u}\,]\!] = \emptyset$.*

Condition 3 implies that '=' is not reflexive. Also, when $v(x) = \mathbf{u}$, atom $x = x$ is false, i.e., $v \notin [\![x = x]\!]$. The following interesting properties for $\mathit{def}(s)$ can also be derived.

Observation 2 *The following conditions hold:*

1. $[\![\mathit{def}(s)]\!] = \bigcup_{d \in \mathcal{D}} [\![s = d]\!]$ *for every term* $s \in \mathcal{T}$,
2. $[\![\mathit{def}(d)]\!] = \mathcal{V}$ *for every domain element* $d \in \mathcal{D}$,
3. $[\![\mathit{def}(x)]\!] = \{ v \in \mathcal{V} \mid v(x) \neq \mathbf{u} \}$ *for every* $x \in \mathcal{X}$,
4. $[\![\mathit{def}(\mathbf{u})]\!] = \emptyset$.

These conditions together imply that constraint terms behave similar to first order terms. That is, we can define a recursive function for valuation of terms and subterms:

Definition 4 (Term valuation) *Given a valuation* $v \in \mathcal{V}$, *an interpretation* $\langle h, t \rangle$ *and a term* $s \in \mathcal{T}$, *we define* $v^{\kappa}_{\langle h,t \rangle} : \mathcal{T} \longrightarrow \mathcal{D}_{\mathbf{u}}$ *as the following function:*

$$v^{\kappa}_{\langle h,t \rangle}(s) \overset{\text{def}}{=} \begin{cases} d & \text{if } v \in [\![\kappa_{\langle h,t \rangle}(s) = d]\!] \text{ with } d \in \mathcal{D}, \\ \mathbf{u} & \text{otherwise} \end{cases}$$

where $\kappa_{\langle h,t \rangle}(s)$ *denotes the basic term that results from replacing each conditional term* s' *in* s *by* $\kappa_{\langle h,t \rangle}(s')$.

For a constraint atom $c \in \mathcal{C}$, we denote by $v^{\kappa}_{\langle h,t \rangle}(c)$ the constraint atom obtained by replacing each occurrence of term s in c by $v^{\kappa}_{\langle h,t \rangle}(s) \in \mathcal{D}_{\mathbf{u}}$. Note that $v^{\kappa}_{\langle h,t \rangle}(c)$ for constraint atom c becomes a syntactic transformation. For example, take as c the following constraint atom:

$$(1|0\colon x = 1) - y \;\geq\; 0$$

which is a slight elaboration of the atom in the body of (4). Suppose that we define $s = (1|0\colon x = 1)$ as a term (remember x, y, 1 and 2 are also elementary terms). Assume also that κ applied to s in this case selects df and take the interpretation $\langle h, t \rangle$ where $h(x) = \mathbf{u}$, $t(x) = 1$ as in previous examples, adding now $h(y) = t(y) = 1$. Then, using df, the conditional term is replaced by 0 in h and by 1 in t. Therefore $v^{\kappa}_{\langle h,t \rangle}(s) = 0$ and $v^{\kappa}_{\langle t,t \rangle}(s) = 1$. Given that the value of y is fixed to 1, we get the basic atoms $v^{\kappa}_{\langle h,t \rangle}(c) = 0 - 1 \geq 0$ and $v^{\kappa}_{\langle t,t \rangle}(c) = 1 - 1 \geq 0$.

Proposition 6 *Given a valuation* v, *an interpretation* $\langle h, t \rangle$, *a selection function* κ, *and an atom* c, *the following two conditions are equivalent:*

1. $v \in [\![\kappa_{\langle h,t \rangle}(c)]\!]$
2. $v \in [\![v^{\kappa}_{\langle h,t \rangle}(c)]\!]$

In other words, we can safely use the term valuation $v^{\kappa}_{\langle h,t \rangle}$ to replace every term for its value in the valuation v wrt the interpreation $\langle h, t \rangle$. Note that, in practice, we chose v to be either h or t. When s is a basic term or $\langle h, t \rangle$ is total, the value returned by $v^{\kappa}_{\langle h,t \rangle}(s)$ does not depend on κ, h or t. For this reason, we just write $v(s)$ in those cases.

Finally, we can establish some relation between the vc- and df-semantics based on how they evaluate constraint terms.

Proposition 7 *Any term* s *and interpretation* $\langle h, t \rangle$ *satisfy that* $h^{vc}_{\langle h,t \rangle}(s) \neq \mathbf{u}$ *implies* $h^{vc}_{\langle h,t \rangle}(s) = h^{\mathit{df}}_{\langle h,t \rangle}(s) = t(s)$.

In other words, if the vc-semantics assigns some value to term s in h, this value is also preserved in t. Moreover, when this is the case the vc- and df-semantics coincide. On the other hand, this preservation property is not satisfied by the df-semantics, as we discussed in the example with the conditional atom in (10).

As mentioned in the introduction, the main motivation to introduce constraint terms is to permit a uniform treatment of different constructs. This is especially relevant for assignments (Cabalar et al. 2016). Intuitively, an assignment of the form $x := s$ is a *directional* construct meaning that variable x takes the value of term s.

Definition 5 (Assignment) *Given a variable* $x \in \mathcal{X}$ *and a term* $s \in \mathcal{T}$ *an* assignment *is an expression of the form* $x := s$ *and stands for the formula*

$$x = s \;\leftarrow\; \mathit{def}(s) \tag{11}$$

Recall that, in (Cabalar et al. 2016), assignments were constructs where s could only be a linear expression. The introduction of terms allows us to generalize the use of assignments to arbitrary terms which, as we can see in following sections, includes both linear expressions and aggregates. The following result provides further intuition relating assignments with grounding in ASP.

Theorem 1 *Consider a formula of the form*

$$x := s \;\leftarrow\; \varphi \tag{12}$$

where $x \in \mathcal{X}$ *is a variable,* $s \in \mathcal{T}$ *a constraint term and* φ *a (sub)formula. Let* Γ *collect the set of formulas:*

$$x = d \;\leftarrow\; \varphi \wedge s = d \tag{13}$$

for every element $d \in \mathcal{D}$ *in the domain. Then,* Γ *and* (12) *have the same* κ*-models.*

In other words, an assignment on the consequent of an implication stands for (the possibly infinite grounding of) the first order formula $\forall Y\,(x = Y \leftarrow \varphi \wedge s = Y)$. Of course, the advantage of assignments consists in the possibility of delegating their evaluation to specialized constraint solvers. For this, such solvers only need to be able to deal with equality constraints. This also implies that grounding is not necessary. Note that, if $\mathcal{D}$ is infinite, then so is Γ.

Aggregates as constraint atoms

Aggregates are expressions that represent a function that groups together a collection of expressions and produces a single value as output. For instance, the expression

$$\mathit{sum}\{\!\!\{\; \mathit{tax}(P) : \mathit{lives}(P, R) \;\}\!\!\} \tag{14}$$

shown in the introduction sums the tax revenue of all persons P that live in some region R. In this section, we restrict ourselves to ground atoms, that is, atoms that may contain constraint variables but no logical variables like P and R. We assume that aggregate atoms with logical variables are a shorthand for their (possibly infinite) ground instantiation. For instance, (14) is a shorthand for the infinite expression of the form $\mathit{sum}\{\!\!\{\alpha_1, \alpha_2, \dots\}\!\!\}$ where each $\alpha_1, \alpha_2, \dots$ is a sequence containing a conditional term of the form $tax(p) : lives(p,r)$ for each pair of domain elements p and r. Intuitively, the variable $lives(p,r)$ is true when the person p lives in the region r and the variable $tax(p)$ is assigned the tax revenue of person p. If p is not a person, $tax(p)$ is undefined, that is, its assigned value is $\mathbf{u}$. As a simpler example, we have the following expression

$$sum\dot{\{}\ 1{:}p,\ 1{:}q,\ 2{:}r\ \dot{\}} \geq 2 \tag{15}$$

which holds if either r holds (regardless of the other variables) or both p and q hold, otherwise. More generally, we allow applying aggregates not only to numerical constants, but also to expressions involving constraint variables. For instance,

$$sum\dot{\{}\ x{:}p,\ y+z{:}q\ \dot{\}} \geq 2 \tag{16}$$

holds whenever any of the following conditions hold:

- $x \geq 2$ and only p holds,
- $y + z \geq 2$ and only q holds, or
- $x + y + z \geq 2$ and both p and q hold.

We also allow aggregate operations that rely on the order of the elements in the collection. For instance, the aggregate

$$\circ\langle\text{“En”, “un”, “lugar”, “de” , “la”, “Mancha”}\rangle = x \tag{17}$$

expresses that x is the string resulting from concatenating all strings occurring between the brackets. That is, it is only satisfied when the value assigned to x is the string “En un lugar de la Mancha”.

Formally, an *aggregate term* is an expression of the form

$$op\langle s_1, s_2, \dots\rangle \tag{18}$$

where op is an *operation symbol* and each $s_i \in \mathcal{T}$ is a term. We say that a term is *aggregate-free* if it contains no aggregate terms and, in the following, we assume that each s_i in (18) is aggregate-free. A *basic aggregate term* is an expression of the form of (18) where each s_i is a basic term. We reserve the notation $\dot{\{}\dots\dot{\}}$ for aggregates whose operation is multi-set based, like sum, and use $\langle\dots\rangle$ in general.

An infinite sequence θ of domain elements $\langle d_0, d_1, \dots\rangle$ can be defined as a mapping $\theta : (\mathbb{N}^+ \to \mathcal{D})$ so that $\theta(i) = d_i$ for all $i \geq 0$. Notice that θ may contain repeated occurrences of the same domain value. We sometimes denote a finite prefix $\theta' = \langle d'_0, \dots, d'_{n-1}\rangle$ of length $n \geq 0$ and use the concatenation $\theta' \cdot \theta$ to yield an infinite sequence defined as expected $\langle d'_0, \dots, d'_{n-1}, d_0, d_1, \dots\rangle$.

Given each aggregate term like (18), we assume there exists an associated fixed operation $\hat{op} : (\mathbb{N}^+ \to \mathcal{D}) \to \mathcal{D}_\mathbf{u}$ assigning a domain value $d \in \mathcal{D}$ (or $\mathbf{u}$) to any infinite sequence of domain values. As an example, in the case of the sum aggregate we get $\hat{sum}\langle d_1, d_2, \dots\rangle = \sum_{i\geq 0} d_i$ as expected. Depending on the domain and the operator, we may sometimes obtain $\mathbf{u}$ as a result. For instance, if $\mathcal{D}$ are the natural numbers and we sum an infinite sequence of 1's, the result of $\hat{sum}\langle 1, 1, 1, \dots\rangle$ is not a natural number and the sum would be undefined $\mathbf{u}$. We say that some $0_{op} \in \mathcal{D}$ is a *neutral element* for op if for all infinite sequence θ and any finite prefix θ' we have $\hat{op}(\theta' \cdot 0_{op} \cdot \theta) = \hat{op}(\theta' \cdot \theta)$. Without loss of generality, we restrict ourselves to operations op that have a neutral element. Otherwise, we can always build an equivalent function with neutral element by adding a new element to the domain.

Definition 6 (Evaluation of a basic aggregate term) *We define the* evaluation $v(A)$ *of a basic aggregate term* A *like* (18) *with respect to a valuation* v *as*

$$v(A) \overset{\text{def}}{=} \begin{cases} \hat{op}(\theta_A) & \textit{if } v(s_i) \neq \mathbf{u}\ \textit{for all } i \geq 1 \\ \mathbf{u} & \textit{otherwise} \end{cases} \tag{19}$$

where $\theta_A : \mathbb{N}^+ \to \mathcal{D}$ *is a function mapping each positive integer* $i \in \mathbb{N}^+$ *to the value* $v(s_i)$.

An *aggregate atom* (or *aggregate* for short) is an expression of the form $A \prec s_0$ where A is an aggregate term, $\prec$ is a relation symbol and s_0 is a basic term. We associate the symbol $\prec$ with some relation $\hat{\prec} \subseteq \mathcal{D} \times \mathcal{D}$ among elements of the domain. The denotation of a basic aggregate atom is then defined as

$$[\![\, A \prec s_0 \,]\!] \overset{\text{def}}{=} \{v \in \mathcal{V} \mid v(A)\ \hat{\prec}\ v(s_0)\}$$

In particular, note that $\prec$ can be the equality symbol. The semantics of conditional aggregates follows directly from the evaluation of conditions introduced in the previous section.

The following result shows how the evaluation of terms introduced in Definition 4 applies to the particular case of aggregate terms.

Proposition 8 (Evaluation of an aggregate term) *We define the* evaluation *of an aggregate term* A *possibly containing conditional terms, with respect to some valuation* $v \in \mathcal{V}$, *some interpretation* $\langle h, t\rangle$ *and a selection function* κ, *as*

$$v^\kappa_{\langle h,t\rangle}(A) \overset{\text{def}}{=} \begin{cases} \hat{op}(\theta^\kappa_{A,\langle h,t\rangle}) & \textit{if } v^\kappa_{\langle h,t\rangle}(s_i) \neq \mathbf{u}\ \textit{for all } i \geq 1 \\ \mathbf{u} & \textit{otherwise} \end{cases}$$

where $\theta^\kappa_{A,\langle h,t\rangle} : \mathbb{N}^+ \to \mathcal{D}$ *is a function mapping each positive integer* $i \in \mathbb{N}^+$ *to the value* $v^\kappa_{\langle h,t\rangle}(s_i)$.

Corollary 2 *Given an aggregate term* A *(possibly with conditional terms), a valuation* $v \in \mathcal{V}$, *some interpretation* $\langle h, t\rangle$ *and a selection function* κ, *we have:*
$\langle h, t\rangle \models A \prec s_0$ *iff* $v^\kappa_{\langle v,t\rangle}(A)\ \hat{\prec}\ v^\kappa_{\langle v,t\rangle}(s_0)$ *for* $v \in \{h, t\}$.

Using neutral elements, we can consider finite aggregates as abbreviations for infinite ones. That is, a finite constraint term of the form

$$op\langle s_1, s_2, \dots, s_n\rangle \tag{20}$$

is an abbreviation for the infinite term

$$op\langle s_1, s_2, \dots, s_n, 0_{op}, 0_{op}, \dots\rangle \tag{21}$$

Treating finite aggregates as an abbreviation allows us to deal with a unique construct for any number of elements and, thus, ensure that aggregate terms with different number of elements are treated in an uniform way.

We also adopt some further conventions for multi-set based aggregates that reflect the syntax of ASP solvers. In particular, a *multi-set aggregate term* is an expression of the form

$$op\dot{\{}\tau_1, \tau_2, \dots\dot{\}} \tag{22}$$

where each τ_i is either a basic term or an expression of the form $s'_i : \varphi_i$ with s'_i a basic term and φ_i a basic formula. Such an expression is understood as an abbreviation for an aggregate term of the form of (18) where each s_i is as follows:

1. $s_i = (\tau_i | 0_{op}\colon \mathit{def}(\tau_i))$ if τ_i is a basic term, and
2. $s_i = (s'_i | 0_{op}\colon \mathit{def}(s'_i) \wedge \varphi_i)$ otherwise.

This allows us to capture the behavior of modern ASP solvers. For instance, the solver *clingo* removes elements that are undefined from the `sum` aggregate and a return value can still be obtained. Now, the semantics of (15) and (16) can be formalized by defining the following function

$$\hat{sum}(\theta) \overset{\text{def}}{=} \sum\{\, \theta(i) \mid i \in \mathbb{N}^+ \text{ and } \theta(i) \in \mathbb{Z} \,\} \qquad (23)$$

where $\theta : \mathbb{N}^+ \to \mathcal{D}$ is a family of domain elements. For $\leq$, we take the usual meaning. Obviously, the neutral element of sum is $0_{sum} = 0$. Note that combining this definition with (19), we get that the sum of an aggregate term is undefined if any of its elements is undefined, otherwise, we get the sum of all integers in the sequence.

Let us illustrate the behavior of aggregates in our setting taking (15) as an example. Note that, following our convention, (15) is a short hand for the atom[5]

$$sum\langle\, (1|0\colon p),\ (1|0\colon q),\ (2|0\colon r),\ 0,\ 0, \dots \rangle \geq 2 \qquad (24)$$

We see that if p, q, r hold in some interpretation, then the left hand side of the inequality evaluates to $\sum\{1, 1, 2\} = 4$ and, the inequality is satisfied. On the other hand, if only p, q hold, we get $\sum\{1, 1\} = 2$ and the inequality is not satisfied. As another example, while evaluating the aggregate term

$$sum\{2,\ 5,\ \text{"hello world"},\ 7\} \qquad (25)$$

the string "hello world" is ignored and the result is just 14.

Beyond arithmetic aggregates, we may also have expressions like (17), which deal with strings. We define $0_\circ$ as the empty string and $\hat{\circ}(\theta)$ as the string $\theta(1)\_\theta(2)\_\dots$ resulting of concatenating all strings in θ.

Aggregates as conditional linear constraints

One important difference between the understanding of aggregates used in this paper and the one studied in (Cabalar et al. 2020) is that the latter directly considers aggregates as abbreviations for conditional linear constraints. This viewpoint is interesting because it allows the use of off-the-shelf CASP solvers to compute aggregates with constraint variables. On the downside, this approach has two drawbacks. First, it is quite different from the usual definition of aggregates in the ASP literature, which makes it difficult to relate to existing approaches in standard (non-constraint) ASP. Second, it is more restrictive as it only permits a particular class of aggregates, namely those using the operation functions `sum`, `count`, `max` and `min`.

The definition we provide in the previous section solves these two issues, but leaves us with the question whether we can use off-the-shelf CASP solver to compute aggregates. In this section, we show that it is possible to translate `sum` aggregates into conditional linear constraints. Thus, affirmatively answering the above question for the vc-semantics. In the next section, we extend this result to an interesting class of theories under the df-semantics. Aggregates with operations `count`, `max` and `min` can be mapped to `sum` ones (Alviano, Faber, and Gebser 2015).

We start by reviewing the definition of *conditional linear constraints* from (Cabalar et al. 2020), but incorporating our notion of term. A *product term* is either an integer $d \in \mathbb{Z}$, a variable $x \in \mathcal{X}$ or an expression of the form $d \cdot x$ where $d \in \mathbb{Z}$ is a domain element and $x \in \mathcal{X}$ is a variable. A *finite basic linear term* is either a product term or an expression of the form $s_1 + \ldots + s_n$ where each s_i is a product term. A *linear term* is an expression of $s_1 + s_2 + \ldots$ where each s_i is either a finite basic linear term or a conditional term of the form $(s'_i | s''_i \colon \varphi_i)$ with s'_i and s''_i finite basic linear terms and φ_i a basic formula. A *linear constraint* is a comparison of the forms $\alpha \leq \beta$, $\alpha < \beta$, $\alpha = \beta$ or $\alpha \neq \beta$ for linear terms α and β. As usual, we write $\alpha \geq \beta$ and $\alpha > \beta$ to stand for $\beta \leq \alpha$ and $\beta < \alpha$, respectively.

We adopt some usual abbreviations. We directly replace the '$+$' symbol by (binary) '$-$' for negative constants and, when clear from the context, omit the '$\cdot$' symbol and parentheses. We do not remove parentheses around conditional expressions. As an example, the expression $-x + \ (3y|2y\colon \varphi) \ - 2z$ stands for $(-1) \cdot x + (3 \cdot y | 2 \cdot y \colon \varphi) + (-2) \cdot z$. Other abbreviations must be handled with care. In particular, we neither remove products of form $0 \cdot x$ nor replace them by 0 (this is because x may be undefined, making the product undefined).

In the rest of the paper, we assume that all integers are part of the domain, that is, $\mathbb{Z} \subseteq \mathcal{D}$. Given a valuation v, the semantics of basic linear constraints is defined inductively.

$$v(d \cdot x) \overset{\text{def}}{=} \begin{cases} d \cdot v(x) & \text{if } v(x) \in \mathbb{Z} \\ \mathbf{u} & \text{otherwise} \end{cases}$$

$$v(s_1 + s_2 + \dots) \overset{\text{def}}{=} \begin{cases} d & \text{if } \forall i \geq 1,\ v(s_i) \in \mathbb{Z} \\ & \text{and } \sum_{i \geq 1} v(s_i) = d \in \mathcal{D} \\ \mathbf{u} & \text{otherwise} \end{cases}$$

The denotation of a basic linear constraint $\alpha \prec \beta$ is given by

$$[\![\, \alpha \prec \beta \,]\!] \overset{\text{def}}{=} \{v \mid v(\alpha), v(\beta) \in \mathbb{Z}, v(\alpha) \prec v(\beta)\}$$

with $\prec$ a relation symbol among $\leq$, $<$, $=$ and $\neq$. In particular, given a linear constraint of the form $\alpha \leq d$ with $\alpha = d_1 \cdot x_1 + \cdots + d_n \cdot x_n$, we have $v \in [\![\, \alpha \leq d \,]\!]$ iff $v(x_i) \neq \mathbf{u}$ for all $1 \leq i \leq n$ and $d \geq \sum_{1 \leq i \leq n} d_i \cdot v(x_i)$.

The semantics of conditional linear constraints is immediately obtained by applying the corresponding evaluation functions. The following results show how the valuation function $v^{\kappa}_{\langle h,t \rangle}$ applies to (conditional) linear terms; and how to use this for evaluating (conditional) linear constraints.

Proposition 9 (Linear term evaluation) *Let $v \in \mathcal{V}$ be a valuation, $\langle h, t \rangle$ be an interpretation and $\alpha = s_1 + s_2 + \ldots$ be a linear term (possibly containing conditional terms). Then,*

$$v^{\kappa}_{\langle h,t \rangle}(\alpha) \ = \ \begin{cases} d & \textit{if } \forall i \geq 1,\ v^{\kappa}_{\langle h,t \rangle}(s_i) \in \mathbb{Z} \\ & \textit{and } \sum_{i \geq 1} v^{\kappa}_{\langle h,t \rangle}(s_i) = d \in \mathcal{D} \\ \mathbf{u} & \textit{otherwise} \end{cases}$$

[5]We dropped the tautologies $\mathit{def}(1)$ and $\mathit{def}(2)$.

Corollary 3 *Given an interpretation $\langle h,t\rangle$ and a linear constraint $\alpha \prec \beta$ (possibly containing conditional terms), we get: $\langle h,t\rangle \models_\kappa \alpha \prec \beta$ iff $v^\kappa_{\langle v,t\rangle}(\alpha) \prec v^\kappa_{\langle v,t\rangle}(\beta)$ for both $v \in \{h,t\}$.*

The following result shows some interesting equivalences.

Proposition 10 *Given an interpretation $\langle h,t\rangle$ and linear terms α and β the following equivalences hold:*

1. *$\langle h,t\rangle \models_\kappa \alpha = \beta$ iff $\langle h,t\rangle \models_\kappa \alpha \leq \beta \wedge \alpha \geq \beta$,*
2. *$\langle h,t\rangle \models_\kappa \alpha < \beta$ iff $\langle h,t\rangle \models_\kappa \alpha \leq \beta \wedge \alpha \neq \beta$,*
3. *$\langle h,t\rangle \models_{vc} \alpha < \beta$ iff $\langle h,t\rangle \models_{vc} \alpha \leq \beta \wedge \neg(\alpha \geq \beta)$,*
4. *$\langle h,t\rangle \models_{vc} \alpha \neq \beta$ iff $\langle h,t\rangle \models_{vc} \alpha < \beta \vee \alpha > \beta$.*

We see that with the vc-semantics, we can define all arithmetic relations in terms of $\leq$, while we need $\leq$ and $\neq$ for the df-semantics. To see that the third equivalence does not hold for the df-semantics, take the interpretation $\langle h,t\rangle$ with $h(x) = \mathbf{u}$ and $t(x) = 1$ and the atom

$$(0|1\colon x = 1) \; < \; 1 \tag{26}$$

Then, with α being the linear term on the left hand side of (26), we get that $\langle h,t\rangle \models_{df} (\alpha \leq 1) \wedge \neg(\alpha \geq 1)$ holds despite of $\langle h,t\rangle \not\models_{df} (\alpha < 1)$. Similarly, for the last equivalence take the same interpretation $\langle h,t\rangle$ and the constraint

$$(0|1\colon x = 1) \; \neq \; (1|0\colon x = 1) \tag{27}$$

Then, with α and β being the linear terms on the left and right hand side of (27), respectively, we get $\langle h,t\rangle \models_{df} (\alpha \neq \beta)$ despite of $\langle h,t\rangle \not\models_{df} (\alpha < \beta) \vee (\alpha > \beta)$.

Let us now show how sum aggregates can be translated into conditional linear constraints. First, we introduce a new constraint atom $int(s)$ whose intuitive meaning is that term s is an integer and whose denotation is given as follows:

$$[\![\, int(s) \,]\!] \quad \stackrel{\text{def}}{=} \quad \bigcup \{ \, [\![\, s = d \,]\!] \mid d \in \mathbb{Z} \, \}$$

Definition 7 (Aggregate to linear term) *Given an aggregate term A of the form (22) with $op = sum$, we associate the linear term $\pi(A) \stackrel{\text{def}}{=} \pi(\tau_1) + \pi(\tau_2) + \ldots$ where $\pi(\tau_i)$ is defined as follows:*

1. *$\pi(\tau_i) \stackrel{\text{def}}{=} (\tau_i|0\colon int(\tau_i))$ if τ_i is a finite basic linear term,*
2. *$\pi(\tau_i) \stackrel{\text{def}}{=} (s_i|0\colon int(s_i) \wedge \varphi_i)$ if τ_i is of the form $s_i : \varphi_i$.*

For a theory Γ, we define $\pi(\Gamma)$ as the result of recursively replacing each aggregate term A by $\pi(A)$ in Γ.

Furthermore, for a selection function κ, we define the selection function $\pi(\kappa)$ given as follows:

1. $\pi(\kappa)(s) = \kappa(s)$ for every occurrence of conditional term s not occurring in any aggregate term A,
2. $\pi(\kappa)(\pi(s)) = \kappa(s)$ for every occurrence of conditional term s occurring in some aggregate term A.

Theorem 2 *For any theory Γ, the κ-(stable) models of Γ and $\pi(\kappa)$-(stable) models of $\pi(\Gamma)$ coincide.*

This means that we can use the techniques developed in (Cabalar et al. 2020) to compute the stable models of theories with aggregate under the vc-semantics.

For instance, (16) becomes the linear constraint

$$(x|0\colon int(x) \wedge p) + (z + y|0\colon int(z + y) \wedge q) \geq 2$$

which holds under the same conditions as (16) does. As a further example, the aggregate term (25) is translated as the linear constraint $\pi(2)+\pi(5)+\pi(\text{“hello world”})+\pi(7)$. For any integer n, we get that $\pi(n) = (n|0\colon int(n))$ is simply equivalent to n because $int(n)$ is a tautology. On the other hand, $\pi(\text{“hello world”})$ is equivalent to 0 because $int(\text{“hello world”})$ is a contradiction. Hence, we get $\pi(2) + \pi(5) + \pi(\text{“hello world”}) + \pi(7) = 2 + 5 + 7 = 14$.

Logic programs

We focus now on a restricted syntax corresponding to logic programs and show that, for stratified occurrences of conditional terms, we can safely exchange vc and df-semantics. A *literal* is either a constraint atom $c \in \mathcal{C}$ or the negation $\neg c$ of one. A *rule* is a formula of the form $H \leftarrow B$ where H is a either an assignment or a disjunction of literals and B is a conjunction of literals called the rule's *head* and *body*, respectively. A theory consisting exclusively of rules is called a *(logic) program*. Further, we adopt the following conventions. For any rule r of form $H \leftarrow B$ we let $H(r)$ and $B(r)$ stand for the set of all literals occurring in H and B, respectively. If H is an assignment $x := s$, we assume that B contains additionally $def(s)$. We denote the set of positive and negative literals in $H(r)$ by $H^+(r) \stackrel{\text{def}}{=} H(r) \cap \mathcal{C}$ and $H^-(r) \stackrel{\text{def}}{=} H(r) \setminus H^+(r)$. We also define $vars^a(c) \stackrel{\text{def}}{=} \{x\}$ if c is an assignment $x := s$ and $vars^a(c) \stackrel{\text{def}}{=} vars(c)$ if c is a constraint atom. Intuitively, $vars^a(c)$ designates all variables assigned by atom c: Only the assigned variable is defined by an assignment and all variables in a constraint atom.

Definition 8 (Conditional term stratification) *A program Π is said to be* stratified on (an occurrence of) a conditional term *$s = (s'|s''\colon \varphi)$, if there is a level mapping $\ell : \mathcal{X} \longrightarrow \mathbb{N}$ satisfying the following conditions for every rule $r \in \Pi$:*

1. *$\ell(x) \geq \ell(y)$ for all variables $x \in vars^a(H^+(r))$ and $y \in vars(H^-(r) \cup B(r))$,*
2. *$\ell(x) = \ell(y)$ for all variables $x, y \in vars^a(H^+(r))$*

plus the following condition for the rule r where s occurs

1. *$\ell(x) > \ell(y)$ for all $x \in vars^a(H^+(r))$ and $y \in var(\varphi)$.*

A program Π is stratified *if it is stratified on all occurrences of conditional terms not occurring in the scope of negation.*

Given selection functions κ, κ' and a distinguished occurrence of some conditional term s, we denote by $\kappa[s \hookleftarrow \kappa']$ the selection functionobtained from κ by replacing the result assigned to s by the one that κ' assigns to it.

Theorem 3 *Let Π be a program stratified on some occurrence of a conditional term s and κ and κ' be two selection functions. Let $\kappa'' = \kappa[s \hookleftarrow \kappa']$. Then, the κ-stable models and the κ''-stable models of Π coincide.*

Theorem 4 *For a stratified program, its κ- and κ'-stable models coincide for any pair of selection functions κ and κ'.*

This means that, for stratified programs, we can use the translation from (Cabalar et al. 2020) to rely on off-the-shelf CASP solvers to compute not only the vc-stable models but also the df-stable models. Furthermore, as our concept of stratification and the translation pertain to distinguished occurrences of conditional terms, it is possible to partially translate non-stratified programs for stratified occurrences.

The proof of Theorem 3 relies on the notions of supported models and splitting sets that we lift from standard ASP to programs with conditional constraints atoms, as stated below. For clarity, we abuse notation and let $H^-(r)$ and $B(r)$ stand for formulas $\bigvee H^-(r)$ and $\bigwedge B(r)$, respectively.

Definition 9 (Supported models) *A variable $x \in \mathcal{X}$ is* supported *wrt a program Π and a valuation v, if there is a rule $r \in \Pi$ and a constraint atom $c \in H^+(r)$ satisfying the following conditions:*

1. $x \in vars^a(c)$,

2. $v \not\models_\kappa c'$ for every $c' \in H^+(r)$ such that $x \notin vars^a(c')$,

3. $v \models_\kappa B(r)$ and $v \not\models_\kappa H^-(r)$.

A model v of a program Π is supported *if every variable that is not undefined is supported wrt. Π and v.*

Proposition 11 *Any stable model of a program is supported.*

Definition 10 (Splitting) *A set of variables $U \subseteq \mathcal{X}$ is a* splitting set *of a program Π, if for any rule r in Π one of the following conditions holds:*

1. $vars(r) \subseteq U$,

2. $vars^a(H^+(r)) \cap U = \emptyset$

We define a splitting *of Π as a pair $\langle B_U(\Pi), T_U(\Pi)\rangle$ satisfying $B_U(\Pi) \cap T_U(\Pi) = \emptyset$, $B_U(\Pi) \cup T_U(\Pi) = \Pi$, all rules in $B_U(\Pi)$ satisfy 1. and all rules in $T_U(\Pi)$ satisfy 2.*

Given a program Π, a splitting set U of Π and a valuation v, we denote by $E_U(\Pi, v)$ the program obtained by replacing each variable $x \in U$ in $T_U(\Pi)$ by $v(x)$. We denote by $\overline{U} \stackrel{\text{def}}{=} \mathcal{X} \setminus U$ the complement of U.

Proposition 12 *Given program Π with splitting set $U \subseteq \mathcal{X}$, a valuation v is a stable model of Π iff $v|_U$ is a stable model of $B_U(\Pi)$ and $v|_{\overline{U}}$ is a stable model of $E_U(\Pi, v|_U)$.*

A generalization of Ferraris' semantics

In this section, we show that, when restricted to *df*-semantics, our approach amounts to a conservative extension of the reduct-based semantics introduced by Ferraris (2011). Under that approach, a classical interpretation is a stable model of a formula if it is a subset minimal classical model of the reduct wrt that interpretation. The reduct of a formula wrt an interpretation is obtained by replacing all maximum subformulas not classically satisfied by the interpretation by $\bot$. We now adapt those notions to HT_C.

Given a denotation $[\![\cdot]\!]$, a valuation t *classically satisfies* a formula φ, written $t \models_{cl} \varphi$, if the following conditions hold:

1. $t \not\models_{cl} \bot$
2. $t \models_{cl} c$ if $t \in [\![\, eval_t(c) \,]\!]$
3. $t \models_{cl} \varphi \wedge \psi$ if $t \models_{cl} \varphi$ and $t \models_{cl} \psi$
4. $t \models_{cl} \varphi \vee \psi$ if $t \models_{cl} \varphi$ or $t \models_{cl} \psi$
5. $t \models_{cl} \varphi \rightarrow \psi$ if $t \not\models_{cl} \varphi$ or $t \models_{cl} \psi$

We say that a valuation v is a *classical model* of theory Γ when $v \models_{cl} \varphi$ for all formulas $\varphi \in \Gamma$.

Observation 3 *For any interpretation $\langle t, t\rangle$, formula φ and selection function κ, we have $\langle t, t\rangle \models_\kappa \varphi$ iff $t \models_{cl} \varphi$*

Definition 11 (Reduct) *The* reduct *of a formula φ wrt an interpretation t, written φ^t, is defined as the expression:*

- *$\bot$ if $t \not\models_{cl} \varphi$ for any formula φ,*
- *$c[s_1^t, s_2^t, \dots]$ if $t \models_{cl} c[s_1, s_2, \dots]$ for any constraint atom $c \in \mathcal{C}$ where $s_1, s_2, \dots$ are all conditional terms in c and $s_i^t \stackrel{\text{def}}{=} (s|s'\!:\varphi^t)$ for each $s_i = (s|s'\!:\varphi)$.*
- *$\varphi_1^t \otimes \varphi_2^t$ if $t \models_{cl} \varphi_1 \otimes \varphi_2$ with $\otimes \in \{\wedge, \vee, \rightarrow\}$*

The reduct of a theory Γ is defined as $\Gamma^t \stackrel{\text{def}}{=} \{\varphi^t \mid \varphi \in \Gamma\}$. A valuation v is called a F-stable model of Γ iff it is a $\subseteq$-minimal model of Γ^t.

An aggregate atom of the form of

$$op\langle\, (s_1|0_{op}\!:\!\varphi_1),\ (s_2|0_{op}\!:\!\varphi_2), \dots \rangle \prec s_0 \qquad (28)$$

can be seen as a generalization of aggregate atoms as defined in (Ferraris 2011) in three ways. First, it permits applying the operation op to both finite or infinite collections of elements. Second, it allows operations op that may or may not depend on the order of the elements in the collection. And third, and more important for our purposes, it allows each s_i to be any expression involving constraint variables rather than just numbers. The following result shows that the application of our reduct to an aggregate of the form of (28) produces a straightforward generalization of Ferraris' reduct.

Proposition 13 *Given an aggregate A of the form (28) and a valuation t, it follows that*

$$A^t = \begin{cases} \bot & \text{if } t \not\models c \\ op\langle (s_1|0_{op}\!:\!\varphi_1^t),\ (s_2|0_{op}\!:\!\varphi_2^t), \dots \rangle \prec s_0 & \text{otherwise} \end{cases}$$

Let us now enunciate the main result of this section.

Theorem 5 *A valuation is a df-stable model of a theory Γ iff it is an F-stable model of Γ.*

Discussion

HT_C is a logic for capturing non-monotonic constraint theories that permits assigning default values to constraint variables. Since ASP is a special case of this logic, it provides a uniform framework for integrating ASP and CP on the same semantic footing. We elaborate on this logic by incorporating *constraint terms*. A notion that allows us to treat linear constraints, conditional expressions and aggregates in a uniform way. In particular, this allows us to introduce assignments for aggregate expressions. We also present a new semantics for conditional expressions in which their result is always defined (*df*) and show that, when combined with an appropriate definition of aggregates, it leads to a generalization of the semantics by Ferraris (2011). Recall that this semantics is the foundation for aggregates in the system *clingo*.

Interestingly, for programs stratified on aggregates, we can translate aggregates using the *df* principle into conditional constraints under the *vicious circle principle*. Then, we can leverage our previous results and translate these constructs into the language of CASP solvers. As a reminder, the fragment covered by the ASP Core 2 semantics (Calimeri et al. 2012) only allows for stratified aggregates.

Ongoing work is directed towards an implementation of a hybrid variant of *clingo* based on the framework developed here. For solving programs with non-stratified aggregates, we are looking into extending the notions of unfounded-sets (Van Gelder, Ross, and Schlipf 1991) and loop formulas (Lin and Zhao 2004) to programs with constraint variables.

References

Alviano, M.; Faber, W.; and Gebser, M. 2015. Rewriting recursive aggregates in answer set programming: Back to monotonicity. *Theory and Practice of Logic Programming* 15(4-5):559–573.

Cabalar, P.; Kaminski, R.; Ostrowski, M.; and Schaub, T. 2016. An ASP semantics for default reasoning with constraints. In Kambhampati, R., ed., *Proceedings of the Twenty-fifth International Joint Conference on Artificial Intelligence (IJCAI'16)*, 1015–1021. IJCAI/AAAI Press.

Cabalar, P.; Fandinno, J.; Fariñas del Cerro, L.; and Pearce, D. 2018. Functional ASP with intensional sets: Application to Gelfond-Zhang aggregates. *Theory and Practice of Logic Programming* 18(3-4):390–405.

Cabalar, P.; Fandinno, J.; Schaub, T.; and Wanko, P. 2020. An asp semantics for constraints involving conditional aggregates. In *Proceedings of the Twenty-fourth European Conference on Artificial Intelligence (ECAI'20)*, to appear.

Calimeri, F.; Faber, W.; Gebser, M.; Ianni, G.; Kaminski, R.; Krennwallner, T.; Leone, N.; Ricca, F.; and Schaub, T. 2012. ASP-Core-2: Input language format.

Ferraris, P. 2011. Logic programs with propositional connectives and aggregates. *ACM Transactions on Computational Logic* 12(4):25.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2019. Multi-shot ASP solving with clingo. *Theory and Practice of Logic Programming* 19(1):27–82.

Gelfond, M., and Zhang, Y. 2019. Vicious circle principle, aggregates, and formation of sets in ASP based languages. *Artificial Intelligence* 275:28–77.

Lierler, Y. 2014. Relating constraint answer set programming languages and algorithms. *Artificial Intelligence* 207:1–22.

Lifschitz, V. 2008. What is answer set programming? In Fox, D., and Gomes, C., eds., *Proceedings of the Twenty-third National Conference on Artificial Intelligence (AAAI'08)*, 1594–1597. AAAI Press.

Lin, F., and Zhao, Y. 2004. ASSAT: computing answer sets of a logic program by SAT solvers. *Artificial Intelligence* 157(1-2):115–137.

Nieuwenhuis, R.; Oliveras, A.; and Tinelli, C. 2006. Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). *Journal of the ACM* 53(6):937–977.

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