# An ASP Semantics for Default Reasoning with Constraints

**381**

## Abstract

We introduce the logic of *Here-and-There with Constraints* in order to capture constraint theories in the non-monotonic setting known from Answer Set Programming (ASP). This allows for assigning default values to constraint variables or to leave them undefined. Also, it provides us with a semantic framework integrating ASP and Constraint Processing in a uniform way. We put some emphasis on logic programs dealing with linear constraints on integer variables, where we further introduce a directional assignment operator. We elaborate upon the formal relation and implementation of these programs in terms of Constraint ASP, sketching an existing system.

## 1 Introduction

Although Answer Set Programming (ASP; [Lifschitz, 2008]) has become a prime candidate for knowledge representation and reasoning, it falls short of succinctly representing variables over large numeric domains. So far, this was addressed by hybridizing ASP with Constraint Processing (CP; [Dechter, 2003]), leading to the subarea of Constraint ASP (CASP; [Lierler, 2014]). In fact, the design of most CASP approaches is inspired by the algorithmic framework of Satisfiability modulo Theories (SMT; [Nieuwenhuis *et al.*, 2006]) and thus leads to hybrid semantics combining non-monotonic aspects of ASP with monotonic ones of CP. This yields an inevitable blind spot, namely, the incapacity of providing defaults for constraint variables (or even leaving them undefined). Such features must be addressed on the ASP side, which brings us back to the aforementioned problem.

We address this dilemma by introducing a new approach that integrates ASP and CP in the uniform semantic framework of the logic of Here-and-There ($HT$; [Heyting, 1930]), extending the Equilibrium Logic [Pearce, 1997] characterization of ASP to theories with constraint atoms. This puts both ASP and CP on the same semantic footing, being non-monotonic in nature. The new logic of *Here-and-There with constraints*, $HT_C$ for short, is built from variables over associated domains, whose valid valuations are determined by the interpretation of constraint atoms. $HT_C$ is not only a proper generalization of $HT$, and hence also ASP, but it also tolerates undefined constraint variables and lets them take default values. Moreover, the logic programming fragment of $HT_C$ also subsumes the CASP approach of [Gebser *et al.*, 2009]. Interestingly, the monotonic nature of constraint variables in CASP can be obtained by adding simple axioms, similar to tertium non datur in $HT$ (or choice rules in ASP). Finally, we elaborate upon the fragment of logic programs with linear constraints on integer variables, $LC$, and introduce directional assignments in rule heads in order to guarantee foundedness in the presence of undefinedness. Furthermore, we develop a translation of $LC$-programs into CASP that forms the backbone of our implementation by means of off-the-shelf CASP solvers.

## 2 Here-and-There with Constraints

In this section, we introduce the logic of *Here-and-There with Constraints*, $HT_C$ for short.

We begin by recalling the definition of a *constraint satisfaction problem* as a triple $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ where $\mathcal{X}$ is a set of variables, $\mathcal{D}$ a domain of values, and $\mathcal{C}$ a set of constraints. Each constraint is a pair $\langle \overline{x}, R \rangle$ where $\overline{x}$ is an $n$-tuple of variables and $R$ an $n$-ary relation on $\mathcal{D}$. A *valuation* of the variables is a function from the set of variables to the domain of values $v : \mathcal{X} \to \mathcal{D}$. A valuation $v$ satisfies a constraint $\langle (x_1, \ldots, x_n), R \rangle$ if $(v(x_1), \ldots, v(x_n)) \in R$. A *solution* of a $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ is a valuation $v$ that satisfies all constraints in $\mathcal{C}$.

The syntax of $HT_C$ starts from a similar signature $\langle \mathcal{X}, \mathcal{D}, \mathcal{A} \rangle$ where, as before, $\mathcal{X}$ are variables and $\mathcal{D}$ domain values, but $\mathcal{A}$ are now *constraint atoms*, or just *atoms* for short. The syntax of constraint atoms is left unspecified but generally they refer to some elements in $\mathcal{X} \cup \mathcal{D}$. Examples of constraint atoms are $x + y \leq 3$, $x = y$, $all\_diff(\{x, y, z\})$, or $x \in \{\boldsymbol{t}, \boldsymbol{f}\}$ where $x, y, z$ are variables and $3, \boldsymbol{t}, \boldsymbol{f}$ are values. We sometimes refer to a subset of variables $\mathcal{P} \subseteq \mathcal{X}$ as *propositions* and let the subset $\{\boldsymbol{t}, \boldsymbol{f}\} \subseteq \mathcal{D}$ of values stand for Boolean truth values. Also, for each proposition $p \in \mathcal{P}$, we include a constraint atom $(p = \boldsymbol{t}) \in \mathcal{A}$ that we call *regular atom* and usually abbreviate by $p$. We also allow for atoms of form $(p = \boldsymbol{f}) \in \mathcal{A}$, standing for the *strong negation* of $p$, and alternatively write them as $\sim p$. A *formula* is any propositional combination of atoms and logical connectives $\bot, \wedge, \vee, \to$. We define negation as $\neg\varphi \overset{\text{def}}{=} \varphi \to \bot$ and double implication as $\varphi \leftrightarrow \psi \overset{\text{def}}{=} (\varphi \to \psi) \wedge (\psi \to \varphi)$.

The semantics of $HT_C$ is defined as follows. A *partial valuation* $v$ is a partial function $v : \mathcal{X} \rightarrow\!\!\!\!\!\!+ \ \mathcal{D}$ assigning domain values to some variables in $\mathcal{X}$. We write $v(x) = \boldsymbol{u}$ to represent that $v(x)$ is undefined. A partial valuation can be alternatively represented as a set $v \subseteq \mathcal{X} \times \mathcal{D}$ that does not contain two different pairs $(x, c)$ and $(x, d)$ with $c \neq d$ for the same variable $x$. We define $\mathcal{V}_{\mathcal{X},\mathcal{D}}$ as the set of all possible partial valuations for $\mathcal{X}$ and $\mathcal{D}$ and remove the subindices when clear from the context. We use letters $v, v'$ to denote elements of $\mathcal{V}$. The closure of a set $\mathcal{S}$ of partial valuations is defined as $\mathcal{S}^{\uparrow} \overset{\text{def}}{=} \{v \mid v \supseteq v', v' \in \mathcal{S}\}$. A set $\mathcal{S}$ of partial valuations is said to be *closed* if $\mathcal{S}^{\uparrow} = \mathcal{S}$. An example of a non-closed set of valuations is $\{v \in \mathcal{V} \mid v(x) = v(y)\}$, since $v = \emptyset$ satisfies $v(x) = v(y) = \boldsymbol{u}$ but its superset $v' = \{(x, 1), (y, 2)\}$ does not satisfy $v'(x) = v'(y)$. On the other hand, $\{v \in \mathcal{V} \mid \boldsymbol{u} \neq v(x) = v(y) \neq \boldsymbol{u}\}$ is closed.

Satisfaction of formulas is defined wrt a fixed function, called *atom denotation*, $[\![ \cdot ]\!] : \mathcal{A} \rightarrow 2^{\mathcal{V}}$ that maps each atom to a closed set of partial valuations. As examples of atom denotations, we have the following:

$$
\begin{aligned}
[\![\, |x - y| = d \,]\!] &\overset{\text{def}}{=} \{v \in \mathcal{V} \mid v(x), v(y) \in \mathbb{Z}, \\
&\qquad\qquad |v(x) - v(y)| = d\} \\
[\![\, x = y \,]\!] &\overset{\text{def}}{=} \{v \in \mathcal{V} \mid \boldsymbol{u} \neq v(x) = v(y) \neq \boldsymbol{u}\} \\
[\![\, x \neq y \,]\!] &\overset{\text{def}}{=} \{v \in \mathcal{V} \mid \boldsymbol{u} \neq v(x) \neq v(y) \neq \boldsymbol{u}\} \\
[\![\, \mathit{all\_diff}(X) \,]\!] &\overset{\text{def}}{=} \{v \in \mathcal{V} \mid \text{for all } x, y \in X \\
&\qquad\qquad \boldsymbol{u} \neq v(x) \neq v(y) \neq \boldsymbol{u}\} \\
[\![\, x \in D \,]\!] &\overset{\text{def}}{=} \{v \in \mathcal{V} \mid v(x) \in D\} \\
[\![\, x \cdot y \doteq 0 \,]\!] &\overset{\text{def}}{=} \{v \in \mathcal{V} \mid v(x) = 0 \text{ or } v(y) = 0\}
\end{aligned}
$$

Note that all above examples satisfy $[\![ A ]\!]^{\uparrow} = [\![ A ]\!]$. Given a denotation $[\![ \cdot ]\!]$ and a partial valuation $v$, we define the set of atoms that hold in $v$ as $At(v) \overset{\text{def}}{=} \{A \in \mathcal{A} \mid v \in [\![ A ]\!]\}$.

**Proposition 1** *Let $v \subseteq v'$ be a pair of partial valuations and $[\![ \cdot ]\!]$ a denotation for atoms in $\mathcal{A}$. Then $At(v) \subseteq At(v')$.*

The denotation for regular atoms is fixed as expected:

$$
\begin{aligned}
[\![ p ]\!] = [\![ p = \boldsymbol{t} ]\!] &\overset{\text{def}}{=} \{v \mid v(p) = \boldsymbol{t}\} \\
[\![ {\sim} p ]\!] = [\![ p = \boldsymbol{f} ]\!] &\overset{\text{def}}{=} \{v \mid v(p) = \boldsymbol{f}\}
\end{aligned}
$$

Let $X = \{x_1, \dots, x_n\}$ be a subset of variables $X \subseteq \mathcal{X}$. A constraint $C = \langle (x_1, \dots, x_n), R \rangle$ can be understood as the following set of partial valuations:

$$\{v \mid \{(x_1, d_1), \dots, (x_n, d_n)\} \subseteq v \text{ and } (d_1, \dots, d_n) \in R\} \ .$$

These are all valuations fixing variables in $X$ to some tuple in $R$ while varying the remaining variables $\mathcal{X} \setminus X$ in all possible ways (including being undefined). Notice that $v(x_i)$ is always defined for all variables in $X$ and any valuation $v$ in the constraint. A set of partial valuations is said to be *strict* if it corresponds to some constraint $C$. Otherwise, it is said to be non-strict. For instance, the denotation $[\![ x = y ]\!]$ provided before is strict because we can represent it as a constraint $\langle (x, y), R \rangle$ where $R = \{(d, d) \mid d \in \mathcal{D}\}$. However, $[\![ x \cdot y \doteq 0 ]\!]$ is non-strict because we may have $v(x) = 0$ but $v(y)$ undefined or the other way around. This cannot be captured by a constraint: we cannot cover this set of valuations as any $\langle (x, y), R \rangle$, $\langle (x), R \rangle$ or $\langle (y), R \rangle$ because we must include both $v_1 = \{(x, 0)\}$, where $y$ is undefined, and $v_2 = \{(y, 0)\}$, where $x$ is undefined. In this paper, we focus on strict denotations for atoms.

An *interpretation* in $HT_C$ is a pair $\langle H, T \rangle$ of partial valuations such that $H \subseteq T$.

**Definition 1 (Satisfaction of formulas)** *Given a fixed denotation $[\![ \cdot ]\!]$, we say that an interpretation $\langle H, T \rangle$* satisfies *a formula $\varphi$, written $\langle H, T \rangle \models \varphi$, when the following recursive conditions hold:*

*(i)* $\langle H, T \rangle \models A$ *iff* $H \in [\![ A ]\!]$

*(ii)* $\bot, \vee, \wedge$ *as usual*

*(iii)* $\langle H, T \rangle \models \varphi \rightarrow \psi$ *iff* $\langle v, T \rangle \not\models \varphi$ *or* $\langle v, T \rangle \models \psi$*, for both* $v = H$ *and* $v = T$*.*

As usual, an interpretation $\langle H, T \rangle$ is a *model* of a theory $\Gamma$, written $\langle H, T \rangle \models \Gamma$, if it satisfies all formulas in $\Gamma$, that is, $\langle H, T \rangle \models \varphi$ for all $\varphi \in \Gamma$. A theory (or a single formula) $\Gamma$ *entails* a formula $\varphi$, written $\Gamma \models \varphi$, when all models of $\Gamma$ are models of $\varphi$. We write $\varphi \equiv \psi$ to represent that $\varphi$ and $\psi$ are *equivalent*, that is, have the same $HT_C$ models.

**Observation 1** *The logic of Here-and-There can be obtained as a case of $HT_C$ with a signature $\langle \mathcal{X}, \mathcal{D}, \mathcal{A} \rangle$ where $\mathcal{X}$ represent propositions, $\mathcal{D} = \{\boldsymbol{t}\}$ and $\mathcal{A} = \mathcal{X}$, understanding each $p \in \mathcal{A}$ as abbreviation of the constraint atom $(p = \boldsymbol{t})$ as explained above. Moreover, this can be generalized to any arbitrary singleton $\mathcal{D} = \{d\}$ and corresponding constraint atoms $(p = d)$ and the relationship still holds.*

The following is an interesting connection between $HT_C$ and $HT$:

**Proposition 2** *Let $\Gamma$ be some $HT_C$ theory for signature $\langle \mathcal{X}, \mathcal{D}, \mathcal{A} \rangle$ and let $\langle H, T \rangle$ be some model of $\Gamma$. Then, $\langle At(H), At(T) \rangle$ is an HT model of $\Gamma$ under signature $\langle \mathcal{A}, \{\boldsymbol{t}\}, \{p = \boldsymbol{t} \mid p \in \mathcal{A}\} \rangle$.*[1]

As a result, we directly derive these properties from $HT$:

**Proposition 3** *For any formula $\varphi$:*

- $\langle H, T \rangle \models \varphi$ *implies* $\langle T, T \rangle \models \varphi$
- $\langle H, T \rangle \models \neg\varphi$ *iff* $\langle T, T \rangle \not\models \varphi$
- *Any tautology in HT is also a tautology in $HT_C$.*

In the light of Proposition 2, one might wonder whether, to capture $HT_C$ semantics, it would suffice to exclusively consider $HT$ theories built with constraint atoms used as propositional variables. This is not the case, since we cannot obtain a similar correspondence in the opposite direction. Namely, not any pair of sets of atoms $H' \subseteq T' \subseteq \mathcal{A}$ necessarily corresponds to an $HT_C$ interpretation $\langle H, T \rangle$ such that $H' = At(H)$ and $T' = At(T)$. As an example, take $H' = \{x = y\} \subset \{x = y, x = 0\} = T'$. Clearly, to obtain $At(T) = T'$ we need $T = \{(x, 0), (y, 0)\}$. Now, the only subset of $T$ that satisfies $x = y$ is $H = T$ itself. But then $At(H) = At(T) = T' \neq H'$.

We sometimes write $T \models \varphi$ to stand for $\langle T, T \rangle \models \varphi$. Extending the equilibrium model definition [Pearce, 1997] to $HT_C$ theories is straightforward.

[1]Or simply $\langle \mathcal{A}, \{\boldsymbol{t}\}, \mathcal{A} \rangle$ by abbreviating all $(p = \boldsymbol{t})$ by $p$.

**Definition 2** *An interpretation $\langle T, T\rangle$ is an* equilibrium model *of a theory $\Gamma$ if $\langle T, T\rangle \models \Gamma$ and there is no $H \subset T$ such that $\langle H, T\rangle \models \Gamma$.*

In this case, we also say that $T$ is a *stable model* of $\Gamma$. Again, if we restrict the signature to $\langle \mathcal{A}, \{\boldsymbol{t}\}, \mathcal{A}\rangle$, we obtain standard equilibrium/stable models.

For logic programming syntax, we use comma ',' and semicolon ';' as alternative representations of $\wedge$ and $\vee$, respectively. Similarly, we write $\varphi \leftarrow \psi$ to stand for $\psi \rightarrow \varphi$, as expected. An $HT_C$-literal is an atom $A$ or its default negation $\neg A$. An $HT_C$ program is a set of rules of the form:

$$L_1; \ldots ; L_n \leftarrow L_{n+1}, \ldots , L_m$$

where each $L_i$ is an $HT_C$-literal.

**Example 1** *For solving the 8-queens puzzle, we define the variables $\mathcal{X} = \{q_1, \ldots , q_8\}$ where $q_i$ represents the column of the queen located at row $i$. We are given some queens already placed and, by default, the first queen should be located at column 1. A possible way to encode this problem is as follows. We use the domain values $\mathcal{D} = \{1, \ldots , 8\}$ and use the atoms $all\_diff(X)$ and $|x - y| = d$ as given above. Then, we specify the problem as the $HT_C$-program $\Pi_1$:*

$$\bot \leftarrow \neg all\_diff(\mathcal{X}) \tag{1}$$

$$\bot \leftarrow |q_i - q_j| = d_{i,j} \tag{2}$$

$$q_1 = 1 \leftarrow \neg(q_1 \neq 1) \tag{3}$$

$$q_k \in \mathcal{D} \tag{4}$$

*where $i, j, k \in \mathcal{D}$, $i \neq j$, $k > 1$ and $d_{i,j}$ is the constant $|i-j|$.*

Without further information, the program $\Pi_1$ yields four solutions corresponding to the possible 8-queens arrangements with $q_1 = 1$. However, if we add the fact $q_1 = 4$, we obtain the 18 possible solutions where queen 1 is located at row 4.

As we can see, constraints can be used to encode default reasoning, such as the default value 1 for variable $q_1$ in the example. This feature of $HT_C$ cannot be represented with the usual semantics for CASP [Gebser *et al.*, 2009] which separates regular ASP atoms (that allow for defaults) from constraint atoms, that only permit monotonic reasoning. As a result, any CASP program that does not contain regular atoms is monotonic. Note the difference wrt $HT_C$ where, due to Observation 1, it is always possible to encode any standard logic program only using constraint variables and picking an arbitrary singleton domain. For instance, the ASP program $\{p \leftarrow \neg q\}$ can be directly encoded as $\{p = 1 \leftarrow \neg(q = 1)\}$ using integer variables instead of Boolean atoms.

Capturing CASP constraints in $HT_C$ can be easily achieved. Take the following pair of axioms:

$$\neg\neg(x = x) \tag{5}$$

$$x = x \vee \neg(x = x) \tag{6}$$

A variable $x$ is said to be *defined* (resp. *rigid*) in a theory $\Gamma$ if the axiom (5) (resp. (6)) is entailed by $\Gamma$.

**Proposition 4** *For any model $\langle H, T\rangle$ of $\Gamma$:*

- *If $x$ is defined in $\Gamma$, then $T(x) \neq \boldsymbol{u}$*
- *If $x$ is rigid in $\Gamma$, then $H(x) = T(x)$*

Intuitively, (5) acts as a constraint forbidding stable models with $x$ undefined. However, $x$ can be undefined in $H$, that is, during models minimization. Thus, a defined variable may be assigned a default value, as we did with $q_1$ in Example 1. On the other hand, (6) forces a monotonic behavior for $x$, so that we can freely choose its value beforehand, including the case in which we decide to leave it undefined. When a variable is both defined and rigid it satisfies (5) and (6), whose conjunction amounts to the axiom $x = x$. This axiom acts as a choice rule allowing to pick any arbitrary value in $\mathcal{D}$ for $x$.

**Theorem 1** *The definition of CASP provided in [Gebser* et al.*, 2009] exactly corresponds to $HT_C$ programs where all variables are defined and rigid.*

In fact, we can apply this same technique (adding axiom $x = x$) to selectively fix a CASP behavior only for some variable $x$. This is analogous to the addition of the ASP choice[2] $p \vee \neg p$ to make proposition $p$ behave classically.

## 3 Logic programs with Linear Constraints

In this section, we focus on a family of constraint atoms for dealing with linear constraints on integer variables, studying some useful syntactic constructions for logic programs with this kind of atoms. A *linear constraint* is a constraint atom of the form $\alpha \leq \beta$ where $\alpha$ and $\beta$ are in their turn linear expressions defined as follows. A *linear expression* $\alpha$ is a sum $t_1 + \cdots + t_n$ where each addend $t_i$ can be a product $d_i \cdot x_i$ or simply a constant $d_i$, being $d_i \in \mathbb{Z}$ and $x_i \in \mathcal{X}$. By $Vars(\alpha)$ we denote the set of variables occurring in $\alpha$ and we sometimes write $Vars(\alpha, \beta) \overset{\text{def}}{=} Vars(\alpha) \cup Vars(\beta)$ when dealing with two linear expressions. A linear constraint $\alpha \leq \beta$ is said to be in normal form if $\beta = d \in \mathbb{Z}$. We adopt some usual abbreviations. We simply write $x_i$ instead of $1 \cdot x_i$ and we directly replace the '+' symbol by '−' for negative constants. Moreover, when clear from the context, we sometimes omit the '$\cdot$' symbol. As an example, $-x+3\,y-2\,z$ stands for $(-1) \cdot x + 3 \cdot y + (-2) \cdot z$. Other abbreviations must be handled with care. In particular, we neither remove products of form $0 \cdot x$ nor replace them by 0.

To define the denotation of a linear constraint, we extend any partial valuation $v$ on integer variables to any arbitrary arithmetic term $t$ in the following way:

$$v(d) \overset{\text{def}}{=} d \qquad \text{if } d \in \mathbb{Z}$$

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

$$v(t_1 \oplus t_2) \overset{\text{def}}{=} \begin{cases} \boldsymbol{u} & \text{if } v(t_1) = \boldsymbol{u} \text{ or } v(t_2) = \boldsymbol{u} \\ v(t_1) \oplus v(t_2) & \text{otherwise} \end{cases}$$

for any variable $x \in \mathcal{X}$ and any operator $\oplus \in \{\cdot, +\}$. As before, we write $v(t) = \boldsymbol{u}$ when $v(t)$ is undefined. In other words, an arithmetic expression is evaluated as usual, except that it is undefined if it contains some undefined subterm (or eventually, some undefined variable).

**Proposition 5** *For any arithmetic expression $t$ and $HT_C$ interpretation $\langle H, T\rangle$, $H(t) \neq \boldsymbol{u}$ implies $H(t) = T(t)$.*

[2] This $HT$-formula is frequently denoted as $\{p\}$ in ASP.

The denotation of a linear constraint $\alpha \leq \beta$ is defined as:

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

This collects interpretations assigning some integer both to $\alpha$ and $\beta$, and additionally $v(\alpha) \leq v(\beta)$. Therefore, $\alpha \leq \beta$ does not hold in interpretations where some variable in $\mathit{Vars}(\alpha, \beta)$ is undefined (or assigned a non-integer value). We can also observe that $[\![ \alpha \leq \beta ]\!]$ is strict, since it can be represented as the constraint $\langle (x_1, \dots, x_n), R\rangle$ with $\mathit{Vars}(\alpha, \beta) = \{x_1, \dots, x_n\}$ and $R$ containing all the $n$-tuples of integer values that assigned to the variables fulfill $v(\alpha) \leq v(\beta)$.

We use some abbreviations: we write $\alpha = \beta$ to stand for the conjunction[3] $\alpha \leq \beta \wedge \beta \leq \alpha$. Given an inequality $A : (\alpha \leq \beta)$, we write $\overline{A}$ to stand for $(\beta < \alpha) \overset{\text{def}}{=} \beta \leq \alpha \wedge \neg(\alpha \leq \beta)$. We also define the formula $\alpha \neq \beta$ as $\alpha < \beta \vee \beta < \alpha$. Notice that $\alpha \neq \beta$ is stronger than $\neg(\alpha = \beta)$ since the former requires $\alpha$ and $\beta$ to have different values (and so, to be both defined), while the latter checks that $\alpha = \beta$ does not hold, and this includes the case in which any of the two is undefined.

One interesting result is that we can fully capture propositional $HT$ and equilibrium logic in $HT_C$ with integer variables and linear constraints. To do so, it suffices to replace each occurrence of a Boolean variable $p$ in a propositional $HT$ theory by the constraint $x_p = 1$ for a corresponding integer variable $x_p$. Then, we get an obvious one-to-one mapping where each assignment $(p, \boldsymbol{t})$ in an $HT$ model corresponds to $(x_p, 1)$ in $HT_C$ and vice versa.

For any linear expression $\alpha$, we define $\mathit{df}\,\alpha \overset{\text{def}}{=} \alpha \leq \alpha$ to stand for "$\alpha$ is defined," that is, $\alpha$ has a value. It is easy to see that $\mathit{df}\,\alpha$ is equivalent to the conjunction $\bigwedge_{x \in \mathit{Vars}(\alpha)} \mathit{df}\,x$. Therefore, if $\alpha$ does not contain integer variables $\mathit{df}\,\alpha = \top$ (the empty conjunction).

Constraints in rule heads must be handled with care because they treat all variables, in principle, in a non-directional way. For instance, imagine we want to assign to $x$ some value in the range from 0 to $y$, assuming that we have assigned some value to $y$ through other rules. Adding the formula $0 \leq x \wedge x \leq y$ would not yield the desired effect because, as we force both constraints to be true, it would also allow for justifying *any arbitrary value* for $y$. To allow for directional assignments, we introduce the following construction. An *assignment* $A$ for variable $x$ is an expression of the form $x := \alpha \mathinner{..} \beta$ (with $\alpha, \beta$ linear expressions) standing for the formula:

$$\neg\neg \mathit{df}\,A \wedge (\mathit{df}\,A \rightarrow \alpha \leq x \wedge x \leq \beta) \tag{7}$$

where $\mathit{df}\,A \overset{\text{def}}{=} \mathit{df}\,\alpha \wedge \mathit{df}\,\beta$. We say that $A$ is *applicable* in $\langle H, T\rangle$ when $\langle H, T\rangle \models \mathit{df}\,A$. We define $\Phi(A)$ to be the non-directional version of assignment $A$, that is, $\Phi(x := \alpha \mathinner{..} \beta) \overset{\text{def}}{=} \alpha \leq x \wedge x \leq \beta$. As we can see, $A$ makes some additional checks regarding the definedness of $\alpha$ and $\beta$ before imposing any condition on $x$. In particular, $(\mathit{df}\,A \rightarrow \alpha \leq x \wedge x \leq \beta)$ guarantees that $\alpha$ and $\beta$ can be used to fix the value of $x$, *but not of variables* in $\alpha$ and $\beta$ themselves. On the other hand, $\neg\neg \mathit{df}\,A$ can be seen as a constraint checking that $\alpha$ and $\beta$ must be eventually defined in the stable model, but through other rule(s) in the program.

When the upper and lower bounds coincide, we just write $(x := \alpha) \overset{\text{def}}{=} (x := \alpha \mathinner{..} \alpha)$, that is, $\neg\neg \mathit{df}\,\alpha \wedge (\mathit{df}\,\alpha \rightarrow x = \alpha)$. Note that, as a result, $\Phi(x := \alpha) = (x = \alpha)$.

The following proposition relates $A$ and its non-directional version, $\Phi(A)$, in some particular cases.

**Proposition 6** *Given an assignment $A = (x := \alpha \mathinner{..} \beta)$ then:*

*(i)* $A \wedge \mathit{df}\,A \equiv \Phi(A)$

*(ii)* $\neg A \equiv \neg\Phi(A)$

In particular, if $A = (x := \alpha \mathinner{..} \beta)$ contains no variables other than the assigned $x$, then $\mathit{df}\,A = \top$ and so $A \equiv \Phi(A)$.

We now define an interesting syntactic subclass of $HT_C$ logic programs. A *linear constraint rule*, or $LC$-rule for short, is a rule of the form:

$$A_1; \dots; A_n \leftarrow B_1, \dots, B_m, \neg B_{m+1}, \dots, \neg B_k \tag{8}$$

with $n \geq 0$ and $k \geq m \geq 0$, where each $A_i$ is an *assignment* and each $B_j$ is a *linear constraint*. For any rule $r$ like (8), we let $\mathit{Head}(r)$ stand for the set of assignments $\{A_1, \dots, A_n\}$ and $\mathit{Body}(r)$ be the set of linear constraints $\{B_1, \dots, B_m, \neg B_{m+1}, \dots, \neg B_k\}$. An $HT_C$ program consisting of $LC$-rules only is called *$LC$-program.*

Notice that an $LC$-rule does not directly correspond to an $HT_C$ program rule since the assignments in the head contain nested implications like (7). However, the following theorem allows us to rewrite any $LC$-rule as a set of $HT_C$ rules, and helps us to illustrate the intuitive behavior of assignments:

**Theorem 2** *A rule $r$ as in* (8) *is equivalent to the conjunction of implications:*

$$\bigvee_{A \in \Delta} \Phi(A) \leftarrow \bigwedge_{A \in \mathit{Body}(r)} A \wedge \bigwedge_{A \in \Delta} \mathit{df}\,A \wedge \bigwedge_{A \in \mathit{Head}(r) \setminus \Delta} \neg\Phi(A)$$

*for all $\Delta \subseteq \mathit{Head}(r)$.*[4]

Due to Proposition 6, each implication in Theorem 2 can be written as a set of $HT_C$ rules, because $\Phi(A)$ is a conjunction in the head $\alpha \leq x \wedge x \leq \beta$ and, by De Morgan, $\neg\Phi(A')$ becomes a disjunction in the body $\neg(\alpha \leq x) \vee \neg(x \leq \beta)$, and both cases can be unfolded in $HT$ into different rules. Let us informally illustrate this result with the following example.

**Example 2** *The $LC$-rule*

$$y := x - 1 \leftarrow \neg(1 \leq z) \tag{9}$$

*corresponds to the set of $HT_C$ rules:*

$$y = x - 1 \leftarrow \neg(1 \leq z), \mathit{df}\,x \tag{10}$$

$$\bot \leftarrow \neg(1 \leq z), \neg(y = x - 1) \tag{11}$$

Suppose our $LC$-program consists of rule (9) only. The intuition is that $\neg(1 \leq z)$ should hold, as $z$ is undefined and we cannot prove $1 \leq z$, but then $y := x - 1$ cannot be fulfilled, since there are no assignments for $x$, so it is left undefined and $x-1$ cannot be evaluated. As a result, we get no stable model.

[3]When we write $x = y$ for two variables, we deal with some syntactic ambiguity wrt '$=$' used as identity constraint atom. In fact, for integer variables, the semantics of both formulas coincide.

[4]A more succinct translation is used in Section 4.

Note how, if we replaced $y := x - 1$ by just $y = x - 1$ in the head, we would get a stable model $T = \{(y, d), (x, d-1)\}$ per each $d \in \mathbb{Z}$ so the rule would also be fixing values for $x$. Looking at the translation in (10)-(11), the behavior of the assignment becomes clearer. As $z$ does not occur in any head, it is left undefined. Variable $x$ occurs in the head of (10), but it depends on $df x$ in the body, and so, this rule cannot be used to provide a founded value for $x$. Thus, $x$ is undefined and $y = x - 1$ is also false, so the constraint (11) becomes applicable, and we get no stable model.

To illustrate non-monotonicity, suppose we add the rule $x := 1$ whose translation from Theorem 2 amounts to the fact $x = 1$. Then, we obtain a unique stable model $\{(x, 1), (y, 0)\}$. Moreover, assume now that together with $x := 1$, we also add the assignment $z := 0..3$. This last version of the program yields four stable models: one with $z = 0$ and $y = 0$ and the other three with $y$ undefined and $z$ varying from 1 to 3.

The next example illustrates the behavior of an $LC$-rule with a disjunction in the head.

**Example 3** *The $LC$-rule:*

$$z := x; t := y \tag{12}$$

*corresponds to the conjunction of the $HT_C$-rules:*

$$z = x; t = y \leftarrow df x, df y \tag{13}$$
$$z = x \leftarrow df x, \neg(t = y) \tag{14}$$
$$t = y \leftarrow df y, \neg(z = x) \tag{15}$$
$$\bot \leftarrow \neg(z = x), \neg(t = y) \tag{16}$$

If we only have (12) in our program, then $x$ and $y$ are undefined and the rule cannot be satisfied – constraint (16) is applicable. If we add, for instance, the assignment $x := 1$, then $y$ is still undefined, but (14) becomes applicable and we get the stable model $\{(x, 1), (z, 1)\}$. Then, if we further add $y := 2$, we obtain the two expected stable models $\{(x, 1), (y, 2), (z, 1)\}$ and $\{(x, 1), (y, 2), (t, 2)\}$. To illustrate how disjunction interacts with positive cycles, let us look at the following variation.

**Example 4** *Take the program containing* (12) *and the rules:*

$$x := 1 \tag{17}$$
$$y := 1 \leftarrow z = 1 \tag{18}$$
$$z := 1 \leftarrow y = 1 \tag{19}$$

If we apply the first disjunct in (12), we get $z = 1$ and then $y = 1$ by (18) leading to the stable model $\{(x, 1), (y, 1), (z, 1)\}$. This is indeed the only stable model of the program. If we tried to apply the second disjunct in (12) instead, we would need to establish a founded value for $y$ first. However, $y$ depends on $z$ which, in its turn, depends on $x$ through the first disjunct of (12). But then the solution $\{(x, 1), (y, 1), (z, 1), (t, 1)\}$ would not be minimal.[5]

We show next that $LC$-programs can be translated into ASP with linear constraints, viz. CASP [Gebser *et al.*, 2009], by introducing some auxiliary propositional variables. CASP semantics was based on the assumption that all constraint variables were defined and rigid, that is, the choice axiom $x = x$ is satisfied for any variable $x$. Let $\boldsymbol{DF}$ stand for the set of choice axioms $x = x$ for all variables in $\mathcal{X}$.

**Proposition 7** *For any linear expression $\alpha$: $\boldsymbol{DF} \models df\,\alpha \equiv \top$*

Let $\Pi$ be an $LC$-program for signature $\Sigma = \langle \mathcal{X}, \mathbb{Z}, \mathcal{A} \rangle$ where $\mathcal{A}$ is a set of linear constraints. We define a set of auxiliary propositions $\mathcal{P} = \{x^\delta \mid x \in \mathcal{X}\}$ that, intuitively, represents the fact that variable $x$ has a defined value in the original program $\Pi$. The translation of $\Pi$ gives a new $HT_C$ program $\tau(\Pi)$ for the extended signature $\tau(\Sigma) = \langle \mathcal{X} \cup \mathcal{P}, \mathbb{Z} \cup \{\boldsymbol{t}\}, \mathcal{A} \cup \mathcal{P} \rangle$. For any linear expression $\alpha$, we write $\alpha^\delta$ to stand for the conjunction of all propositions $y^\delta$ for all $y \in \mathit{Vars}(\alpha)$ and $(\alpha \leq \beta)^\delta$ to stand for the conjunction $\alpha^\delta \wedge \beta^\delta$. Using this notation, the translation of a linear constraint $A$ is the formula $\tau(A) \stackrel{\text{def}}{=} A \wedge A^\delta$. Intuitively, due to the choice axiom $\boldsymbol{DF}$ (applied only on $\mathcal{X}$), $A$ can hold due to an arbitrary assignment of variable values, but $\mathcal{A}^\delta$ guarantees that all variables have been assigned a founded value wrt program $\Pi$. Notice that the translation of $df\,\alpha$ corresponds to $\tau(df\,\alpha) = \tau(\alpha \leq \alpha) = \alpha \leq \alpha \wedge \alpha^\delta$ and, under the assumption $\boldsymbol{DF}$, the latter is equivalent to $\alpha^\delta$ (Proposition 7). For any arbitrary formula $\phi$, $\tau(\phi)$ stands for the replacement of every constraint atom $A$ in $\phi$ by $\tau(A)$. The translation of an $LC$-program $\Pi$ corresponds to the set of formulas $\tau(\Pi) \stackrel{\text{def}}{=} \{\tau(r) \mid r \in \Pi\}$.

As we see below, the models of the translation $\tau(\Pi)$ are isomorphic to the original models of $\Pi$. Thus, we can apply $\tau$ on rules of the form (8) or on their decomposition through Theorem 2. As an example, if we apply the translation on the decomposition of (12) as (13)-(16), we obtain, after some minor simplifications, the rules:

$$z = x \wedge z^\delta; t = y \wedge t^\delta \leftarrow x^\delta, y^\delta \tag{20}$$
$$z = x \wedge z^\delta \leftarrow x^\delta, \neg(t = y \wedge t^\delta \wedge y^\delta) \tag{21}$$
$$t = y \wedge t^\delta \leftarrow y^\delta, \neg(z = x \wedge z^\delta \wedge x^\delta) \tag{22}$$
$$\bot \leftarrow \neg(z = x \wedge z^\delta \wedge x^\delta), \neg(t = y \wedge t^\delta \wedge y^\delta) \tag{23}$$

Given a valuation $v$ for the extended signature $\tau(\Sigma)$, we define its corresponding "defined" subset in signature $\Sigma$ as $v|_\delta \stackrel{\text{def}}{=} \{(x, d) \in v \mid (x^\delta, \boldsymbol{t}) \in v\}$.

**Proposition 8** *For a pair of partial valuations $H \subseteq T$ for signature $\tau(\Sigma)$, then $H|_\delta \subseteq T|_\delta$.*

**Lemma 1** *Let $\langle H, T \rangle$ be an $HT_C$ interpretation for signature $\tau(\Sigma)$. Then, for any constraint atom $A \in \mathcal{A}$, $\langle H, T \rangle \models \tau(A)$ iff $\langle H|_\delta, T|_\delta \rangle \models A$.*

By a simple application of structural induction, we get:

**Corollary 1** *Let $\langle H, T \rangle$ be an $HT_C$ interpretation for signature $\tau(\Sigma)$. Then, for any formula $\varphi$, $\langle H, T \rangle \models \tau(\varphi)$ iff $\langle H|_\delta, T|_\delta \rangle \models \varphi$.*

**Theorem 3 (Soundness)** *Let $T$ be a stable model of $\tau(\Pi) \cup \boldsymbol{DF}$. Then, $T|_\delta$ is a stable model of $LC$-program $\Pi$.*

**Theorem 4 (Completeness)** *Let $T$ be a stable model of $LC$-program $\Pi$. Then, any $T'$ such that $T' \models \boldsymbol{DF}$ and $T'|_\delta = T$ is a stable model of $\tau(\Pi) \cup \boldsymbol{DF}$.*

[5]As in standard ASP, stable models of a positive $HT_C$ program are always minimal wrt set inclusion.

In other words, each stable model $T$ of $\Pi$ is in one-to-one correspondence to a class of stable models $T'$ of $\tau(\Pi) \cup \boldsymbol{DF}$ that coincide with $T$ in the valuation of its defined variables, making $x^\delta$ true for all of them, and letting the rest of variables vary freely. For instance, as we saw above, the program $\Pi$ consisting of (12) plus the facts $x := 1$ and $y := 2$ has two stable models $T_1 = \{(x, 1), (y, 2), (z, 1)\}$ and $T_2 = \{(x, 1), (y, 2), (t, 2)\}$. Then, $\tau(\Pi)$ includes the formulas (20)-(23) plus the translation of the facts, viz. $x = 1 \wedge x^\delta$ and $y = 2 \wedge y^\delta$. Additionally, $\boldsymbol{DF}$ includes the axioms $x = x$, $y = y$ and $z = z$. The resulting translation $\tau(\Pi) \wedge \boldsymbol{DF}$ yields two sets of stable models: one of the form

$$\{(x, 1), (y, 2), (z, 1), (x^\delta, \boldsymbol{t}), (y^\delta, \boldsymbol{t}), (z^\delta, \boldsymbol{t}), (t, d)\}$$

for any $d \in \mathcal{D}$ and a second one such as

$$\{(x, 1), (y, 2), (t, 2), (x^\delta, \boldsymbol{t}), (y^\delta, \boldsymbol{t}), (t^\delta, \boldsymbol{t}), (z, d)\}$$

again, for any $d \in \mathcal{D}$.

## 4 An $LC$-solver implementation

We implemented our approach (see [LC2CASP, 2016]) as an extension of the CASP solver CLINGCON [Ostrowski and Schaub, 2012]. Our system computes the stable models of an $LC$-program by implementing a polynomial-size variant of the translation described in the previous section. This is accomplished by using auxiliary atoms to avoid the exponential blow-up in Theorem 2 (similar to [Tseitin, 1968]).

In the input language, rule heads are formed by means of the functor `&assign`. More precisely, a disjunctive head $A_1; \ldots; A_n$ as in (8) is represented as '`&assign {` $A_1$`;`$\ldots$`;`$A_n$ `}`'. Similarly, linear expressions are formed using the `&sum` functor. A linear constraint of form '$\alpha_1 + \cdots + \alpha_n \prec \beta$' is written as '`&sum {` $\alpha_1$`;`$\ldots$`;`$\alpha_n$ `}` $\prec \beta$', where $\prec$ is among `<=`, `=`, `>=`, `<`, `>`, or `!=`. Moreover, the language contains an all-different constraint, `&distinct`, as well as a `&show` and `&minimize` directive with the same meaning as in ASP yet applied to linear expressions. As with ASP, undefined variables are not shown (eg. $t$ and $z$ above, respectively); also, they do not contribute to minimization.

For illustration, consider the $HT_C$-program in (1) to (4) expressed as an $LC$-program:

```
1   n(1..8).
2   :- not &distinct { q(X) : n(X) }.
3   :- &sum { q(X); -q(Y) } = X-Y, n(X), n(Y), X != Y.
4   :- &sum { q(X); -q(Y) } = Y-X, n(X), n(Y), X != Y.
5   &assign { q(1) := 1 } :- not &sum { q(1) } != 1.
6   &assign { q(X) := 1..n } :- n(X), X > 1.
```

Note that atoms, like `n(X)` and `X > 1`, are Boolean propositions, mixed with constraint atoms. The above $LC$-program has 4 stable models, all assigning `1` to `q(1)` according to the default expressed in Line 5. However, once '`&assign { q(1) := 4 }.`' is added, the default is overwritten, and we obtain 18 models, yet all assigning `4` to `q(1)`.

## 5 Discussion

We introduced the logic $HT_C$ in order to capture constraint theories in the non-monotonic setting known from ASP. As a result, $HT_C$ allows for assigning default values to constraint variables or to leave them undefined. To the best of our knowledge, $HT_C$ constitutes the first logical account of non-monotonic constraint theories. Since $HT$ and thus also ASP constitute special cases of $HT_C$, we obtain a uniform framework integrating ASP and CP on the same semantic footing. In view of this, we particularly elaborated on the $HT_C$ fragment of $LC$-programs dealing with linear constraints on integer variables. A central concept is that of assignments (in rule heads) because they are the only way to attribute values to constraint variables – unassigned variables stay undefined.

Our approach is different from traditional CASP [Baselice *et al.*, 2005; Balduccini, 2009; Gebser *et al.*, 2009], where logic programs are hybridized with constraint atoms having standard monotonic CP semantics. In such approaches, constraint atoms in rule heads are merely shortcuts for the complementary body literal. Rather, the monotonic CP semantics assigns each variable all feasible values. In fact, we have identified the $HT_C$ fragment corresponding to the approach of [Gebser *et al.*, 2009] and pinpointed the axioms characterizing the aforementioned feature. Although we have not proven it, the result should also extend in a slightly different form to the approaches in [Baselice *et al.*, 2005; Balduccini, 2009] due to their close correspondence to [Gebser *et al.*, 2009] established in [Lierler, 2014]. A noteworthy exception among CASP approaches is Bound Founded ASP [Aziz *et al.*, 2013] that imports the notion of non-circular value derivations into CP. Informally, constraints can have a distinguished variable (akin to a head) over a totally ordered domain. The singular value of a lower-bound[6] variable is the smallest derivable value or the smallest domain element. This yields also a non-monotonic approach that comprises ASP as a special case. However, it remains future work to identify the fragment of $HT_C$ that captures this approach and its notion of value minimization.

Our semantics captures a fragment of ASP with partial functions [Cabalar, 2011; Balduccini, 2012] where constraint variables correspond to 0-ary functions. This fragment is expressive enough to cover the general case, since arbitrarily nested partial functions can be reduced to the 0-ary case by a process called *flattening* [Cabalar, 2011] or *unfolding* [Bartholomew and Lee, 2013]. Moreover, our approach extends functional ASP by generalizing equality among terms to arbitrary relations. In this paper, we have focused on linear constraints, but other extensions will be studied in the future.

For implementing the fragment of $LC$-programs, we have devised a translation into CASP programs in accord with [Gebser *et al.*, 2009] and shown its soundness and completeness. The key role in this translation is played by propositions warranting the non-circularity of constraint assignments. Although our system uses CLINGCON as back-end, our translational approach applies also to other CASP solvers sharing the same semantics. Our system along with several examples as well as proofs of theorems is available at [LC2CASP, 2016].

[6]And analogously for upper-bound variables.

## References

[Aziz *et al.*, 2013] R. Aziz, G. Chu, and P. Stuckey. Stable model semantics for founded bounds. *Theory and Practice of Logic Programming*, 13(4-5):517–532, 2013.

[Balduccini, 2009] M. Balduccini. Representing constraint satisfaction problems in answer set programming. In W. Faber and J. Lee, editors, *Proceedings of the Second Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP'09)*, pages 16–30, 2009.

[Balduccini, 2012] Marcello Balduccini. A "conservative" approach to extending answer set programming with non-herbrand functions. In Esra Erdem, Joohyung Lee, Yuliya Lierler, and David Pearce, editors, *Correct Reasoning*, pages 24–39. Springer-Verlag, 2012.

[Bartholomew and Lee, 2013] M. Bartholomew and J. Lee. On the stable model semantics for intensional functions. *Theory and Practice of Logic Programming*, 13(4-5):863–876, 2013.

[Baselice *et al.*, 2005] S. Baselice, P. Bonatti, and M. Gelfond. Towards an integration of answer set and constraint solving. In M. Gabbrielli and G. Gupta, editors, *Proceedings of the Twenty-first International Conference on Logic Programming (ICLP'05)*, volume 3668 of *Lecture Notes in Computer Science*, pages 52–66. Springer-Verlag, 2005.

[Cabalar *et al.*, 2005] Pedro Cabalar, David Pearce, and Agustín Valverde. Reducing propositional theories in equilibrium logic to logic programs. In *Proc. of the 12th Portuguese Conference on Artificial Intelligence (EPIA'05)*, volume 3808 of *Lecture Notes in Computer Science*, pages 4–17. Springer-Verlag, 2005.

[Cabalar, 2011] P. Cabalar. Functional answer set programming. *Theory and Practice of Logic Programming*, 11(2-3):203–233, 2011.

[Dechter, 2003] R. Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, 2003.

[Gebser *et al.*, 2009] M. Gebser, M. Ostrowski, and T. Schaub. Constraint answer set solving. In P. Hill and D. Warren, editors, *Proceedings of the Twenty-fifth International Conference on Logic Programming (ICLP'09)*, volume 5649 of *Lecture Notes in Computer Science*, pages 235–249. Springer-Verlag, 2009.

[Heyting, 1930] A. Heyting. Die formalen Regeln der intuitionistischen Logik. In *Sitzungsberichte der Preussischen Akademie der Wissenschaften*, page 42–56. 1930. Reprint in Logik-Texte: Kommentierte Auswahl zur Geschichte der Modernen Logik, Akademie-Verlag, 1986.

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

[Lifschitz *et al.*, 1999] Vladimir Lifschitz, Lappoon R. Tang, and Hudson Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25(3–4):369–389, 1999.

[Lifschitz *et al.*, 2001] Vladimir Lifschitz, David Pearce, and Agustín Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2(4):526–541, October 2001.

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

[Nieuwenhuis *et al.*, 2006] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. 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, 2006.

[Ostrowski and Schaub, 2012] M. Ostrowski and T. Schaub. ASP modulo CSP: The clingcon system. *Theory and Practice of Logic Programming*, 12(4-5):485–503, 2012.

[Pearce, 1997] D. Pearce. A new logical characterisation of stable models and answer sets. In J. Dix, L. Pereira, and T. Przymusinski, editors, *Proceedings of the Sixth International Workshop on Non-Monotonic Extensions of Logic Programming (NMELP'96)*, volume 1216 of *Lecture Notes in Computer Science*, pages 57–70. Springer-Verlag, 1997.

[LC2CASP, 2016] `github.com/lc2casp/lc2casp`, 2016.

[Tseitin, 1968] G. Tseitin. On the complexity of derivation in the propositional calculus. *Zapiski nauchnykh seminarov LOMI*, 8:234–259, 1968.

## Appendix. Proofs

**Proof of Theorem 1**. To avoid entering into formal details from CASP [Gebser *et al.*, 2009], we just provide an informal proof sketch. The definition of CASP answer sets starts from some total interpretation $v$ that fixes a (defined) value for all variables beforehand. Then, given the constraint logic program $\Pi$, we can define the reduct $\Pi^v$ that removes rules in which constraint literals in the body do not hold with respect to $v$, and remove the rest of constraint literals (true w.r.t. $v$). CASP answer sets correspond to standard answer sets of $\Pi^v$ (which is an ASP program without constraints), that is, standard equilibrium models of $\Pi^v$.

As we saw, in the case of $HT_C$, the addition of axiom $\boldsymbol{DF}$ makes that all constraint variables satisfy $H(x) = T(x) \neq \boldsymbol{u}$. Thus, any constraint atom will have the same truth value both in $H$ and $T$ worlds. In other words, if we fix the values of each constraint atom $A$ in $T$, we can replace its occurrence in $\Pi$ by $\top$ or $\bot$ depending on whether $A$ holds or not in $T$, and the resulting program will still be satisfied by $\langle H, T\rangle$. As a result, it is not difficult to see that any $HT$ model of $\Pi^v$ is in one-to-one correspondence with an $HT_C$ model (under axiom $\boldsymbol{DF}$) of $\Pi$ where $H(x) = T(x) = v(x)$ for any constraint variable $x$. Since the equilibrium minimization in $HT_C$ for $\Pi$ is not affected by rigid variables, in this case, it essentially amounts to the equilibrium minimization in $HT$ for $\Pi^v$. $\square$

**Proof of Proposition 1**. For any atom $A \in \mathit{At}(v)$ we have $v \in [\![ A ]\!]$. As $[\![ A ]\!]$ is closed and $v' \supseteq v$, we conclude $v' \in [\![ A ]\!]$, and thus, $A \in \mathit{At}(v')$. $\square$

**Proof of Proposition 2**. First, note that $\langle \mathit{At}(H), \mathit{At}(T)\rangle$ is a well-formed $HT$ interpretation since $\mathit{At}(H) \subseteq \mathit{At}(T)$ by $H \subseteq T$ and Proposition 1. Then, the result follows from the fact that (i) in Definition 1 can be rephrased as $A \in \mathit{At}(H)$ whereas the satisfaction conditions for the rest of operators remain unchanged. $\square$

**Proof of Proposition 5**. This is straightforward, since $H$ only differs from $T$ in that the former may leave some variables undefined. But $H(t) \neq \boldsymbol{u}$ implies that all variables in $\mathit{Vars}(t)$ are defined in $H$, and so, they have the same value in $T$, implying $T(t) = H(t)$. $\square$

**Proof of Proposition 6**. For proving (i), notice that the expression $A \wedge \mathit{df}\, A$ corresponds to:

$$\neg\neg \mathit{df}\, A \wedge (\mathit{df}\, A \rightarrow \alpha \leq x \wedge x \leq \beta) \wedge \mathit{df}\, A$$

but since $\varphi \models \neg\neg\varphi$ and $\varphi \wedge (\varphi \rightarrow \psi) \equiv \varphi \wedge \psi$ in $HT$, the formula above is equivalent to:

$$\alpha \leq x \wedge x \leq \beta \wedge \mathit{df}\, A$$

Finally, as $\alpha \leq x \wedge x \leq \beta \models \mathit{df}\, A$ we can remove the conjunct $\mathit{df}\, A$ above.

For (ii) we have:

$$\begin{array}{rcl} \neg A & \equiv & \neg(\neg\neg \mathit{df}\, A \wedge (\mathit{df}\, A \rightarrow \alpha \leq x \wedge x \leq \beta) \\ & \equiv & \neg\neg\neg \mathit{df}\, A \vee \neg\neg \mathit{df}\, A \wedge \neg(\alpha \leq x \wedge x \leq \beta) \\ & \equiv & \neg \mathit{df}\, A \vee \neg(\alpha \leq x \wedge x \leq \beta) \end{array}$$

But, as $(\alpha \leq x \wedge x \leq \beta) \models \mathit{df} A$, we conclude $\neg \mathit{df}\, A \models \neg(\alpha \leq x \wedge x \leq \beta)$ and so the formula above is equivalent to $\neg(\alpha \leq x \wedge x \leq \beta)$. $\square$

**Lemma 2** *The following are valid HT-equivalences:*

$$\begin{array}{rl} \gamma \vee (\varphi \rightarrow \psi) \equiv (\varphi \rightarrow \psi \vee \gamma) & \\ \wedge\, (\neg\psi \rightarrow \neg\varphi \vee \gamma) & (24) \\ (\varphi \rightarrow (\psi \rightarrow \gamma)) \equiv (\varphi \wedge \psi \rightarrow \gamma) & (25) \\ (\varphi \rightarrow \psi \wedge \gamma) \equiv (\varphi \rightarrow \psi) \wedge (\varphi \rightarrow \gamma) & (26) \\ \gamma \vee \neg\neg\varphi \equiv \neg\varphi \rightarrow \gamma & (27) \\ \gamma \vee \neg\neg\varphi \wedge (\varphi \rightarrow \psi) \equiv (\varphi \rightarrow \psi \vee \gamma) & \\ \wedge\, (\neg\psi \rightarrow \gamma) \wedge (\neg\varphi \rightarrow \gamma) & (28) \\ & (29) \end{array}$$

*Proof.* (24) and (25) are subcases of transformation (R5) in [Cabalar *et al.*, 2005] whereas (26) and (27) respectively correspond to (i) and (iv) from Proposition 6 (iv) in [Lifschitz *et al.*, 1999] for nested expressions, which are valid in HT [Lifschitz *et al.*, 2001]. For (28) we apply De Morgan to the left hand side obtaining:

$$\begin{array}{rl} & (\gamma \vee \neg\neg\varphi) \wedge (\gamma \vee (\varphi \rightarrow \psi)) \\ \equiv & (\gamma \vee \neg\neg\varphi) \wedge (\varphi \rightarrow \gamma \vee \psi) \wedge (\neg\psi \rightarrow \gamma \vee \neg\varphi) \end{array}$$

where we applied (24) in the second conjunct. By (27), we can replace the first conjunct by $\neg\varphi \rightarrow \gamma$ but then, in the presence of this last conjunct, the consequent of the last implication $\gamma \vee \neg\varphi$ can be replaced by $\gamma$ obtaining:

$$(\neg\varphi \rightarrow \gamma) \wedge (\varphi \rightarrow \gamma \vee \psi) \wedge (\neg\psi \rightarrow \gamma) \qquad \square$$

**Lemma 3** *Let $A$ be a constraint atom of the form $x := \alpha \mathrel{..} \beta$. Then $\gamma \vee A$ is equivalent to:*

$$(\mathit{df}\, A \rightarrow \Phi(A) \vee \gamma) \wedge (\neg\Phi(A) \rightarrow \gamma)$$

*Proof.* By (7), $A$ corresponds to the formula $\neg\neg \mathit{df}\, A \wedge (\mathit{df}\, A \rightarrow \Phi(A))$. This formula follows the pattern of the left hand side of (28), making the replacements $\varphi$ by $\mathit{df}\, A$ and $\psi$ by $\Phi(A)$. As a result, we obtain that $A$ is equivalent to:

$$(\mathit{df}\, A \rightarrow \Phi(A) \vee \gamma) \wedge (\neg \mathit{df}\, A \rightarrow \gamma) \wedge (\neg\Phi(A) \rightarrow \gamma)$$

But now, we observe that $\Phi(A) \models \mathit{df}\, A$ since satisfying $\alpha \leq x \wedge x \leq \beta$ always implies satisfying $\mathit{df}\alpha$ and $\mathit{df}\beta$. Since HT satisfies contraposition, $\neg \mathit{df}\, A \models \neg\Phi(A)$ and so $(\neg \mathit{df}\, A \rightarrow \gamma)$ is subsumed by $(\neg\Phi \rightarrow \gamma)$, so that we can remove the former. $\square$

**Proof of Theorem 2**. For any $i = 0, \dots, n$ let $\mathit{Hd}_i$ to stand for the set $\{A_1, \dots, A_i\}$. Note that when $i = 0$, $\mathit{Hd}_i = \emptyset$. We prove that by induction $i$ that (8) is equivalent to the set of rules $S_i$ defined as:

$$\begin{array}{rr} \gamma_i \vee \displaystyle\bigvee_{A \in \Delta} \Phi(A) \leftarrow & \\ \displaystyle\bigwedge_{A \in \mathit{Body}(r)} A \wedge \bigwedge_{A \in \Delta} \mathit{df}\, A \wedge \bigwedge_{A' \in \mathit{Hd}_i \setminus \Delta} \neg\Phi(A') & (30) \end{array}$$

for all $\Delta \subseteq \mathit{Hd}_i$, where $\gamma_i$ stands for $A_{i+1} \vee \cdots \vee A_n$. For $i = 0$ we have that $\gamma_i = \mathit{Head}(r)$ and $\mathit{Hd}_i = \emptyset$ so its unique subset is $\Delta = \emptyset$ and the expression above trivially amounts to (8) (empty disjunctions and conjunctions respectively amount to $\bot$ and $\top$, as usual). For the inductive step, assume it holds for $0 \leq i < n$ and we want to prove it for $i+1$. Take any rule like (30) in $S_i$ for some fixed $\Delta \subseteq Hd_i$. Since $i < n$, $\gamma_i = A_{i+1} \vee \gamma_{i+1}$. If we apply Lemma 3 on the head of (30) taking $\gamma = \gamma_{i+1} \vee \bigvee_{A \in \Delta} \Phi(A)$ and $A = A_{i+1}$ we obtain the conjunction of the two implications:

$$\begin{aligned} \Phi(A_i) \vee \gamma_{i+1} \vee \bigvee_{A \in \Delta} \Phi(A) &\leftarrow \quad df A_i \\ \gamma_{i+1} \vee \bigvee_{A \in \Delta} \Phi(A) &\leftarrow \quad \neg A_i \end{aligned}$$

in the head of the rule. Now, using (26) to split the conjunction in the head into two different implications, and (25) to remove nested implications, we get the pair of rules:

$$\gamma_{i+1} \vee \bigvee_{A \in \Delta \cup \{A_i\}} \Phi(A) \leftarrow \bigwedge_{A \in Body(r)} A \wedge \bigwedge_{A \in \Delta \cup \{A_i\}} df A \wedge \bigwedge_{A' \in Hd_i \setminus \Delta} \neg \Phi(A') \quad (31)$$

$$\gamma_{i+1} \vee \bigvee_{A \in \Delta} \Phi(A) \leftarrow \bigwedge_{A \in Body(r)} A \wedge \bigwedge_{A \in \Delta} df A \wedge \bigwedge_{A' \in \{A_i\} \cup Hd_i \setminus \Delta} \neg \Phi(A') \quad (32)$$

It is not difficult to see that these two rules belong to $S_{i+1}$ and respectively correspond to the subsets $\Delta \cup \{A_i\}$ and $\Delta$ of $Hd_{i+1}$ – notice that $Hd_{i+1} \setminus (\Delta \cup \{A_i\}) = Hd_i \setminus \Delta$. Moreover, for any rule in $S_{i+1}$ fixing some $\Delta' \subseteq Hd_{i+1}$, we may find the corresponding rule in $S_i$ with $\Delta = \Delta' \setminus \{A_i\}$ so that splitting the latter generates the former. Therefore, using this splitting for each rule in $S_i$ we get exactly all rules in $S_{i+1}$, and the inductive step is proved.

Finally, it simply remains to observe that the set of rules in the enunciate of the Theorem corresponds to the case $i = n$, where $\gamma_i = \top$ (the empty disjunction) and $Hd_i = Head(r)$. $\square$

**Proof of Proposition 8**. If $(x, d) \in H|_\delta$, by definition of $H|_\delta$, we have both $(x, d) \in H$ and $(x^\delta, \boldsymbol{t}) \in H$. As $H \subseteq T$, we also have these two pairs in $T$, but then, by definition of $T|_\delta$, we conclude $(x, d) \in T|_\delta$.

**Proof of Lemma 1**. First, note that $\langle H|_\delta, T|_\delta \rangle$ is a well-formed $HT_C$ interpretation due to Proposition 8. Second, observe that $\langle H, T \rangle \models \tau(A)$ is equivalent to $\langle H, T \rangle \models A \wedge A^\delta$. For the left to right direction, $\langle H, T \rangle \models \tau(A) = A \wedge A^\delta$ $\langle H, T \rangle \models A^\delta$ implies $(x^\delta, \boldsymbol{t}) \in H$ for all $x \in Vars(A)$. But then, for that set of variables, $H|_\delta(x) = H(x)$ and so $\langle H, T \rangle \models A$ implies $\langle H|_\delta, T|_\delta \rangle \models A$. For the right to left direction, if $\langle H|_\delta, T|_\delta \rangle \models A$ then all variables in $A$ are defined in $H|_\delta$, that is, for all $x \in Vars(A)$ there is some $d$ such that $(x, d) \in H|_\delta$. But, by construction of $H|_\delta$, $(x, d) \in H|_\delta$ implies $(x^\delta, \boldsymbol{t}) \in H$ and $(x, d) \in H$. So $\langle H|_\delta, T|_\delta \rangle \models A$ implies $\langle H, T \rangle \models A \wedge A^\delta = \tau(A)$.

**Proof of Theorem 3**. Suppose $T$ is a stable model of $\tau(\Pi) \cup$ $\boldsymbol{DF}$. By Lemma 1, $T \models \tau(\Pi)$ implies $T|_\delta \models \Pi$. Suppose we had some $H' \subset T|_\delta$ such that $\langle H', T|_\delta \rangle \models \Pi$. Then we can build the partial valuation $H = H' \cup \{(x^\delta, \boldsymbol{t}) \mid H'(x) \neq \boldsymbol{u}\} \cup \{(x, d) \in T \mid H'(x) \neq \boldsymbol{u}\}$. Notice that $H(x) = T(x)$ for all $x \in \mathcal{X}$, since $H' \subset T|_\delta \subseteq T$ repeats the values of some variables from $T$, while $H$ collects the rest of values from $T$ for those undefined variables in $H'$ too. As a result, $\langle H, T \rangle \models$ $\boldsymbol{DF}$. On the other hand, $H \subset T$ because there are some $x^\delta$ atoms in $T$ that do not belong to $H$ (those corresponding to variables $x$ with a value in $H'$ but not in $T|_\delta$). Moreover, it is easy to see that $H' = H|_\delta$. But then, by Lemma 1, $\langle H', T|_\delta \rangle = \langle H|_\delta, T|_\delta \rangle \models \Pi$ implies $\langle H, T \rangle \models \tau(\Pi)$, which contradicts that $T$ is a stable model of $\tau(\Pi) \cup \boldsymbol{DF}$. $\square$

**Proof of Theorem 4**. Suppose $T$ is stable model of $\Pi$ but there is some $T'$ such that $T'|_\delta = T$ and $T' \models \boldsymbol{DF}$ that is not stable model of $\tau(\Pi) \cup \boldsymbol{DF}$. Since $T$ is a stable model of $\Pi$ we have $T = T'|_\delta \models \Pi$ and, by Lemma 1, $T' \models \tau(\Pi)$. Assume there is some $H' \subset T'$ such that $\langle H', T' \rangle \models \tau(\Pi) \cup \boldsymbol{DF}$. Due to axiom $\boldsymbol{DF}$, $H'(x) = T'(x)$ for all $x \in \mathcal{X}$, and so, $H' \subset T'$ implies that there exists some $x^\delta$ such that $(x^\delta, \boldsymbol{t}) \in T' \setminus H'$. As a result, $H'|_\delta \subset T'|_\delta = T$. On the other hand, by Lemma 1 again, $\langle H', T' \rangle \models \tau(\Pi)$ implies $\langle H'|_\delta, T'|_\delta \rangle \models \Pi$, i.e., $\langle H'|_\delta, T \rangle \models \Pi$. But this contradicts the fact that $T$ was a stable model of $\Pi$. $\square$