TPLP: Page 1–22. © The Author(s), 2026. Published by Cambridge University Press. This is an Open Access article, distributed under the terms of the Creative Commons Attribution licence (https://creativecommons.org/licenses/by/4.0/), which permits unrestricted re-use, distribution and reproduction, provided the original article is properly cited.

doi:10.1017/S1471068426100647

*Strong Equivalence in Answer Set Programming with Constraints**

PEDRO CABALAR
University of A Coruña, Spain
(*e-mail:* pedro.cabalar@udc.es)

JORGE FANDINNO
Computer Science, University of Nebraska Omaha, USA
(*e-mail:* jfandinno@unomaha.edu)

TORSTEN SCHAUB
Institut für Informatik und Computational Science, University of Potsdam, Germany
(*e-mail:* torsten@cs.uni-potsdam.de)

PHILIPP WANKO
Potassco Solutions GmbH, Germany
(*e-mail:* philipp.wanko@potassco.com)

submitted 1 April 2025 UTC; revised 10 June 2026 UTC; accepted 12 June 2026 UTC

Abstract

We investigate the concept of strong equivalence within the extended framework of Answer Set Programming with constraints. Two groups of rules are considered strongly equivalent if, informally speaking, they have the same meaning in any context. We demonstrate that, under certain assumptions, strong equivalence between rule sets in this extended setting can be precisely characterized by their equivalence in the logic of Here-and-There with constraints. Furthermore, we present a translation from the language of several `clingo`-based answer set solvers that handle constraints into the language of Here-and-There with constraints. This translation enables us to leverage the logic of Here-and-There to reason about strong equivalence within the context of these solvers. We also explore the computational complexity of determining strong equivalence in this context.

KEYWORDS: theory, knowledge representation and nonmonotonic reasoning

1 Introduction

Many real-world applications have a heterogeneous nature that can only be effectively handled by combining different types of constraints. This is commonly addressed by

* This work was supported by grant PID2023-148531NB-I00 funded by Spanish Ministry MCIU/AEI/10.13039/501100011033, funds FEDER, EU, by the National Science Foundation under Grant No. 95-3101-0060-402 and the CAREER award 2338635, USA, and by DFG grant SCHA 550/15, Germany. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

https://doi.org/10.1017/S1471068426100647 Published online by Cambridge University Press

hybrid solving technology, most successfully in the area of Satisfiability modulo Theories (SMT; Nieuwenhuis *et al*. 2006) with neighboring areas such as Answer Set Programming (ASP; Lifschitz 2008) following suit. However, this increased expressiveness introduces a critical challenge: How can we determine whether modifications to a heterogeneous specification preserve its semantic meaning, in view of the intricate interplay of different constraint types? This is even more severe in ASP since two logic programs may be equivalent (in the sense of having the same answer sets) and yet, replacing one by the other in a larger context may produce different effects, due to the nonmonotonic nature of ASP semantics. For this, one generally needs a stronger concept of equivalence that guarantees that, informally speaking, two expressions have the same meaning in any context. Formally, this is referred to as *strong equivalence* (Lifschitz *et al*. 2001). Such properties are important because they can help us simplify parts of a logic program in a modular way, without examining its other parts.

This paper delves into the question of strong equivalence within the extended framework of Constraint Answer Set Programming (CASP; Lierler 2023), where the integration of linear arithmetic constraints over integers introduces new difficulties. To address this challenge, we harness the logic of Here-and-There with constraints (HT_c; Cabalar *et al*. 2016, 2020a, 2020b), a powerful formalism that allows us to precisely characterize extensions of ASP with external theories over arbitrary domains. By employing HT_c, we aim to develop a formal method for analyzing strong equivalence in CASP, contributing to a deeper understanding of program optimization in this increasingly important paradigm.

As an example, consider the following very simple ASP program with constraints in the language of the hybrid solver `clingcon` (Banbara *et al*. 2017):

```
:- not a,  &sum{s} >= 120.
a :-  &sum{s} > 100.
```

The Boolean atom `a` is true when an alarm is set in the car and `s` is an integer variable representing the speed of the car. The general form of a linear constraint in `clingcon` allows adding multiple arguments in the theory atom `&sum{...}` but, in this case, we just use it to check the value of the integer variable `s`. The first rule tells us that if the alarm is not set, we cannot choose speeds exceeding 120 km/h. The second one states that if the speed is greater than 100 km/h, then the alarm must be set. We are interested in answering the question of whether the first rule is redundant in the presence of the second rule. In terms of strong equivalence, we want to know if the pair of rules together are strongly equivalent to the second rule alone. As we discuss in Section 3, this is indeed the case.

Our main result is a characterization of strong equivalence in the context of ASP with constraints, given in Section 8. The proof of this result is based on two intermediate results that are of independent interest: A characterization of strong equivalence in the context of HT_c developed in Section 3 and an answer set preserving transformation of logic programs with constraints into HT_c-theories given in Section 4. We also study the computational complexity of deciding strong equivalence in this context in Section 9. Finally, Section 10 concludes the paper. Regarding the preliminary material, we split it into three separate sections closer to where the material is needed: Section 2 introduces HT_c, which is used throughout the paper. Section 4 introduces the syntax and semantics

https://doi.org/10.1017/S1471068426100647 Published online by Cambridge University Press

of ASP with constraint, which is used in Section 5 to introduce a new, simplified definition of theory stable model that is equivalent to the original definition under widely accepted assumptions that are inherent to most hybrid solvers. The simplified definition is used in the remainder of the paper. Section 6 reviews structured and compositional theories that are used in the transformation of logic programs with constraints into HT_c-theories in Section 7.

2 Review: The logic of here-and-there with constraints

The logic of *Here-and-There with constraints* (HT_c) along with its equilibrium models provides logical foundations for constraint satisfaction problems (CSPs) in the setting of ASP. Its approach follows the one of the traditional Boolean Logic of Here-and-There (Heyting 1930) and its nonmonotonic extension of Equilibrium Logic (Pearce 1997, 2006). In HT_c, a CSP is expressed as a triple $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, also called *signature*, where $\mathcal{X}$ is a set of *variables* over some non-empty *domain* $\mathcal{D}$ and $\mathcal{C}$ is a set of *constraint atoms*. A constraint atom provides an abstract way to relate values of variables and constants according to the atom's semantics. Most useful constraint atoms have a structured syntax, but in the general case, we may simply consider them as strings.[1] For instance, linear equations are expressions of the form "$x + y = 4$", where x and y are variables from $\mathcal{X}$ and 4 is a constant representing some element from $\mathcal{D}$.

Variables can be assigned some value from $\mathcal{D}$ or left *undefined*. For the latter, we use the special symbol $\mathbf{u} \notin \mathcal{D}$ and the extended domain $\mathcal{D}_\mathbf{u} \stackrel{\text{def}}{=} \mathcal{D} \cup \{\mathbf{u}\}$. The set $vars(c) \subseteq \mathcal{X}$ collects all variables occurring in constraint atom c.

A *valuation* v over $\mathcal{X}, \mathcal{D}$ is a set of pairs $x \mapsto d$ where $x \in \mathcal{X}$ and $d \in \mathcal{D}$ such that for each $x \in \mathcal{X}$ there is at most one pair $x \mapsto d$ in v. The restriction of v to a set of variables $Z \subseteq \mathcal{X}$, denoted $v|_Z$, is the set of pairs $x \mapsto d$ in v such that $x \in Z$. We define the *domain* of a valuation v as the set of variables that have a defined value, namely,

$$dom(v) \stackrel{\text{def}}{=} \{x \in \mathcal{X} \mid x \mapsto d \in v \text{ for some } d \in \mathcal{D}\}.$$

We sometimes use function notation for writing valuations and write $v(x) = d$ to denote that $x \mapsto d$ is in v, and $v(x) = \mathbf{u}$ to denote that there is no pair $x \mapsto d$ in v for any $d \in \mathcal{D}$. Using this notation, $v \subseteq v'$ means that $v(x) \in \mathcal{D}$ implies $v'(x) = v(x)$ for all $x \in \mathcal{X}$. We also allow for applying a valuation v to fixed domain values, and so extend their type to $v : \mathcal{X} \cup \mathcal{D}_\mathbf{u} \to \mathcal{D}_\mathbf{u}$ by fixing $v(d) = d$ for any $d \in \mathcal{D}_\mathbf{u}$. The semantics of constraint atoms is defined in HT_c via *denotations*, which are functions of form $[\![\cdot]\!] : \mathcal{C} \to 2^\mathcal{V}$, mapping each constraint atom to a set of valuations. Following Cabalar *et al.* (2020a), we assume in the sequel that they satisfy the following properties for all $c \in \mathcal{C}$, $x \in \mathcal{X}$, and $v, v' \in \mathcal{V}$:

1. $v \in [\![c]\!]$ and $v \subseteq v'$ imply $v' \in [\![c]\!]$,
2. $v \in [\![c]\!]$ implies $v \in [\![c[x/v(x)]]\!]$,
3. if $v(x) = v'(x)$ for all $x \in vars(c)$ then $v \in [\![c]\!]$ iff $v' \in [\![c]\!]$.

[1] We just need that there is a way to identify variables occurring in a constraint atom, and a way to replace variables by their values. This can be implemented as an interface that the developer of a specific theory must provide.

https://doi.org/10.1017/S1471068426100647 Published online by Cambridge University Press

where $c[s/s']$ is the syntactic replacement in c of subexpression s by s'. We also assume that $c[x/d] \in \mathcal{C}$ for any constraint atom $c \in \mathcal{C}$ containing x, for variable $x \in \mathcal{X}$ and $d \in \mathcal{D}_{\mathbf{u}}$. That is, replacing a variable by any element of the extended domain results in a syntactically valid constraint atom. Intuitively, Condition 1 makes constraint atoms behave monotonically with respect to definedness. Condition 2 stipulates that denotations respect the role of variables as placeholders for values, that is, replacing variables by their assigned values does not change how an expression is evaluated. Condition 3 asserts that the denotation of c is fixed by combinations of values for $vars(c)$; other variables may freely vary.

A formula φ over signature $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ is defined as

$$\varphi ::= \bot \mid c \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \quad \text{where } c \in \mathcal{C}.$$

We define $\top$ as $\bot \rightarrow \bot$ and $\neg\varphi$ as $\varphi \rightarrow \bot$ for any formula φ. We let $vars(\varphi)$ stand for the set of variables in $\mathcal{X}$ occurring in all constraint atoms in formula φ. An HT_c-*theory* is a set of formulas.

In HT_c, 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 $h = t$. Given a denotation $[\![\cdot]\!]$, an interpretation $\langle h, t \rangle$ *satisfies* a formula φ, written $\langle h, t \rangle \models \varphi$, if

1. $\langle h, t \rangle \models c$ if $h \in [\![c]\!]$
2. $\langle h, t \rangle \models \varphi \wedge \psi$ if $\langle h, t \rangle \models \varphi$ and $\langle h, t \rangle \models \psi$
3. $\langle h, t \rangle \models \varphi \vee \psi$ if $\langle h, t \rangle \models \varphi$ or $\langle h, t \rangle \models \psi$
4. $\langle h, t \rangle \models \varphi \rightarrow \psi$ if $\langle w, t \rangle \not\models \varphi$ or $\langle w, t \rangle \models \psi$ for every $w \in \{h, t\}$

We say that an interpretation $\langle h, t \rangle$ is a model of a theory Γ, written $\langle h, t \rangle \models \Gamma$, when $\langle h, t \rangle \models \varphi$ for every $\varphi \in \Gamma$. We write $\Gamma \models \Gamma'$ when every model of Γ is also a model of Γ'. We write $\Gamma \equiv \Gamma'$ if Γ and Γ' have the same models. We omit braces whenever a theory is a singleton.

Proposition 1 (Cabalar et al. 2016, Proposition 3).
For any formula φ, we have

1. $\langle h, t \rangle \models \varphi$ *implies* $\langle t, t \rangle \models \varphi$,
2. $\langle h, t \rangle \models \neg\varphi$ *iff* $\langle t, t \rangle \not\models \varphi$, *and*
3. *any tautology in Here-and-There (Heyting 1930) is also a tautology in* HT_c.

The first item reflects the well-known persistence property in constructive logics. The second one tells us that negation is only evaluated in the there world. The third item allows us to derive the strong equivalence of two expressions in HT_c from their equivalence in HT when treating constraint atoms monolithically, though more equivalences can usually be derived using HT_c.

HT_c makes few assumptions about the syntactic form or the semantics of constraint atoms. In the current paper, however, we introduce a specific kind of constraint atom that is useful later on for some of the formal results. Given a subset $\mathcal{D}' \subseteq \mathcal{D}$, we define the associated[2] constraint atom $x : \mathcal{D}'$ with denotation:

[2] Technically, $\mathcal{D}'$ must not be thought as an element of the atom syntax but as part of the atom name, that is we have a different atom for each $\mathcal{D}' \subseteq \mathcal{D}$.

https://doi.org/10.1017/S1471068426100647 Published online by Cambridge University Press

$$[\![x : \mathcal{D}']\!] \stackrel{\text{def}}{=} \{v \in \mathcal{V}^{\mathcal{X},\mathcal{D}} \mid v(x) \in \mathcal{D}'\}$$

and $vars(x : \mathcal{D}') = \{x\}$, that is, $x : \mathcal{D}'$ asserts that x has some value in subdomain $\mathcal{D}'$. Note that $v(x) \in \mathcal{D}'$ implies that x is defined in v, since $\mathbf{u} \notin \mathcal{D} \supseteq \mathcal{D}'$. We use the abbreviation $def(x)$ to stand for $x : \mathcal{D}$, so that this atom holds iff x has some value, that is it is not undefined, or $v(x) \neq \mathbf{u}$.

The nonmonotonic extension of HT_c is defined in terms of equilibrium models, being minimal models in HT_c in the following sense.

Definition 1 (Equilibrium/Stable model).
A (total) interpretation $\langle t, t\rangle$ is an *equilibrium model* of a theory Γ, if $\langle t, t\rangle \models \Gamma$ and there is no $h \subset t$ such that $\langle h, t\rangle \models \Gamma$. If $\langle t, t\rangle$ is an equilibrium model of Γ, then we say that t is a *stable model* of Γ.

As detailed by Cabalar *et al.* (2016), the original logic of Here-and-There can be obtained as a special case of HT_c with a signature $\langle \mathcal{X}, \mathcal{D}, \mathcal{C}\rangle$, where $\mathcal{X} = \mathcal{A}$ represents logical propositions, the domain $\mathcal{D} = \{\mathbf{t}\}$ contains a unique value (standing for "true") and the set of constraint atoms $\mathcal{C} = \mathcal{A}$ coincides with the set of propositions. In this way, each logical proposition $\mathtt{a}$ becomes both a constraint atom $\mathtt{a} \in \mathcal{C}$ and a homonymous variable $a \in \mathcal{X}$ so that $vars(\mathtt{a}) = \{a\}$ (we use different fonts for the same name to stress the different role). The denotation of a propositional atom is fixed to

$$[\![\mathtt{a}]\!] \stackrel{\text{def}}{=} [\![a : \{\mathbf{t}\}]\!] = \{v \in \mathcal{V}^{\mathcal{X},\mathcal{D}} \mid v(a) = \mathbf{t}\}$$

In the following, we refer to constraint atoms with this denotation as *propositional atoms*. Moreover, we can establish a one-to-one correspondence between any propositional interpretation, represented as a set X of propositional atoms, and the valuation v that assigns $\mathbf{t}$ to all members of X, that is, $X = \{\mathtt{a} \mid v(a) = \mathbf{t}\}$. Note that a false atom, viz. $\mathtt{a} \notin X$, is actually undefined in the valuation, viz. $v(a) = \mathbf{u}$, since having no value is the default situation.[3] Once this correspondence is established, it is easy to see that the definition of equilibrium and stable models in Definition 1 collapses to their standard definition for propositional theories (and also, logic programs in ASP).

Treating logical propositions as constraint atoms allows us not only to capture standard ASP but also to combine propositional atoms with other constraint atoms in a homogeneous way, even when we deal with a larger domain $\mathcal{D} \supset \{\mathbf{t}\}$ such as, for instance, $\mathcal{D} = \mathbb{Z} \cup \{\mathbf{t}\}$. In this setting, we may observe that while $[\![\mathtt{a}]\!] \subseteq [\![def(a)]\!]$ (i.e., if the atom has value $\mathbf{t}$, it is obviously defined), the opposite $[\![def(a)]\!] \subseteq [\![\mathtt{a}]\!]$ does not necessarily hold. For instance, we may have now some valuation $v(a) = 7$, and so $v \in [\![def(a)]\!]$, but $v \notin [\![\mathtt{a}]\!]$. For this reason, we assume the inclusion of the following axiom

$$def(a) \rightarrow \mathtt{a} \tag{1}$$

to enforce $[\![def(a)]\!] = [\![\mathtt{a}]\!]$ for each propositional atom $\mathtt{a}$. This forces propositional atoms to be either true or undefined. It does not constrain non-propositional atoms from taking any value in the domain.

[3] Allowing the assignment of a second truth value $\mathbf{f} \in \mathcal{D}$ for variable a would rather correspond to the explicit negation $-\mathtt{a}$ of the propositional atom.

https://doi.org/10.1017/S1471068426100647 Published online by Cambridge University Press

Finally, one more possibility we may consider in HT_c is treating all constraint atoms as propositional atoms, so that we do not inspect their meaning in terms of an external theory but only consider their truth as propositions.

Definition 2 (Regular stable model).
Let Γ be an HT_c theory over signature $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$. A set of atoms $X \subseteq \mathcal{C}$ is a *regular stable model* of Γ if the valuation $t = \{\mathtt{a} \mapsto \mathbf{t} \mid \mathtt{a} \in X\}$ is a stable model of Γ over signature $\langle \mathcal{C}, \{\mathbf{t}\}, \mathcal{C} \rangle$ while fixing the denotation $[\![\mathtt{a}]\!] \stackrel{\text{def}}{=} [\![a : \{\mathbf{t}\}]\!]$ for every $\mathtt{a} \in \mathcal{C}$.

In other words, regular stable models are the result of considering an HT_c-theory Γ as a propositional HT-theory where constraint atoms are treated as logical propositions. This definition is useful in defining the semantics of logic programs with constraints according to `clingo` (see Section 4).

3 Strong equivalence in $\mathbf{HT}_c$

One of the most important applications of HT is its use in ASP for equivalent transformations among different programs or fragments of programs. In general, if we want to safely replace program P by Q, it does not suffice to check that their sets of stable models coincide, because the semantics of ASP programs cannot be figured out by looking at single rules (or groups of rules) in isolation. We can easily extrapolate this concept to HT_c as follows:

Definition 3 (Strong equivalence).
HT_c-Theories Γ and Γ' are *strongly equivalent* when $\Gamma \cup \Delta$ and $\Gamma' \cup \Delta$ have the same stable models for any arbitrary HT_c-theory Δ.

Theory Δ is sometimes called a *context*, so that strong equivalence guarantees that Γ can be replaced by Γ' (and vice versa) in any context. An important property of HT proved by Lifschitz *et al.* (2001) is that two logic programs P and Q are strongly equivalent if and only if they are equivalent in HT.

Accordingly, we may wonder whether a similar result holds for HT_c. The following theorem is an immediate result: the equivalence $\Gamma \equiv \Gamma'$ in HT_c implies that HT_c-theories Γ and Γ' are strongly equivalent.

Theorem 1.
If HT_c-theories Γ and Γ' are equivalent in HT_c, that is, $\Gamma \equiv \Gamma'$, then Γ and Γ' are strongly equivalent.

Proof.
Assume $\Gamma \equiv \Gamma'$ and take any arbitrary theory Δ. Then, Γ and Γ' have the same models and, as a result, $\Gamma \cup \Delta$ and $\Gamma' \cup \Delta$ have also the same models. Since equilibrium models are a selection among HT_c models, their equilibrium and stable models also coincide. □

For illustration, let us use Theorem 1 to prove the strong equivalence of two simple HT_c-theories. Consider the following two formulas, similar to the ones from the introductory section:

$$\bot \leftarrow \neg \mathtt{a} \wedge s \geq 120 \tag{2}$$

https://doi.org/10.1017/S1471068426100647 Published online by Cambridge University Press

$$\mathtt{a} \leftarrow s > 100 \tag{3}$$

In fact, we show in Section 7 that these two formulas are an essential part of the HT_c-based translation of the logic program mentioned in the introduction. Let us show that $\Gamma = \{(2), (3)\}$ is strongly equivalent to $\Gamma' = \{(3)\}$. By Theorem 1, a sufficient condition is showing that $\Gamma \equiv \Gamma'$ and, since $\Gamma' \subseteq \Gamma$, it suffices to show that (3) $\rightarrow$ (2) is an HT_c-tautology. The following property is useful to prove this result.

Let $\varphi[c/\psi]$ be the uniform replacement of a constraint atom c occurring in formula φ by a formula ψ, and $\varphi[c]$ be the formula φ with a distinguished occurrence of constraint atom c. We show below that HT_c satisfies the rule of uniform substitution.

Proposition 2.
If $\varphi[c]$ is an HT_c tautology then $\varphi[c/\psi]$ is an HT_c-tautology.

Proof.
As a proof sketch, simply observe that if the original formula $\varphi[c]$ is a tautology, it is satisfied for any possible combination of satisfactions for c in h and in t. Each time we replace c by some formula ψ in a uniform way for some interpretation $\langle h, t\rangle$, this corresponds to one of these possible truth combinations for atom c, and so $\varphi[c/\psi]$ is also satisfied. □

Let us now resume our example. The HT-tautology

$$(\alpha \rightarrow \beta) \rightarrow ((\beta \rightarrow \gamma) \rightarrow \neg(\neg\gamma \wedge \alpha))$$

is also an HT_c-tautology by Proposition 1. By applying Proposition 2 to this HT_c-tautology with substitution $\alpha \mapsto s \geq 120$, $\beta \mapsto s > 100$, and $\gamma \mapsto \mathtt{a}$, we obtain the following HT_c-tautology:

$$(s \geq 120 \rightarrow s > 100) \rightarrow ((3) \rightarrow (2))$$

To show that (3) $\rightarrow$ (2) is an HT_c-tautology and, thus, Γ and Γ' are strongly equivalent, it suffices to show that the antecedent of this implication is an HT_c-tautology. This immediately follows once we assume the usual semantics of linear inequalities and, thus, that our denotation satisfies

$$[\![\, s \geq 120 \,]\!] \subseteq [\![\, s > 100 \,]\!].$$

As expected, the result may not hold if we assume that these two constraint atoms have a different meaning than the one they have for linear inequalities.

This illustrates how we can use Theorem 1 to prove the strong equivalence of two HT_c-theories. To prove that HT_c-equivalence is also a necessary condition for strong equivalence, we depend on the form of the context theory Δ. For instance, in the case in which Γ and Γ' are regular ASP programs, it is well-known that if we restrict the form of Δ to sets of facts, we obtain a weaker concept called *uniform equivalence*, that may hold even if Γ and Γ' do not have the same HT models. In the case of HT_c, the variability in the possible context theory Δ is much higher since the syntax and semantics of constraint atoms are very general, and few assumptions can be made on them. Yet, if our theory accepts at least a constraint atom $def(x)$ in $\mathcal{C}$ for each $x \in \mathcal{X}$, then we can

https://doi.org/10.1017/S1471068426100647 Published online by Cambridge University Press

use this construct (in the context theory Δ) to prove the other direction of the strong equivalence characterization.

Theorem 2.
If HT_c-theories Γ and Γ' are strongly equivalent, then they are equivalent in HT_c, that is, $\Gamma \equiv \Gamma'$.

Proof.
We proceed by contraposition, that is, proving that $\Gamma \not\equiv \Gamma'$ implies that Γ and Γ' are not strongly equivalent. To this aim, we build some theory Δ such that $\Gamma \cup \Delta$ and $\Gamma' \cup \Delta$ have different stable models. Without loss of generality, suppose there exists some model $\langle h, t\rangle \models \Gamma$ but $\langle h, t\rangle \not\models \Gamma'$. Note that, by persistence, $\langle t, t\rangle \models \Gamma$, but we do not know whether $\langle t, t\rangle \models \Gamma'$ or not, so we separate the proof in two cases.

Case 1. Suppose first that $\langle t, t\rangle \not\models \Gamma'$. Let us build the theory

$$\Delta = \{\, def(x) \mid x \in \mathcal{X}, t(x) \neq \mathbf{u}\}$$

that exclusively consists of constraint atoms. We can easily see that $\langle t, t\rangle \models \Delta$ because Δ collects precisely those $def(x)$ for which $t(x) \neq \mathbf{u}$ and so, $t \in [\![\, def(x) \,]\!]$ trivially. As a result, $\langle t, t\rangle \models \Gamma \cup \Delta$ and, to prove that $\langle t, t\rangle$ is in equilibrium, suppose we have a smaller $v \subset t$ satisfying $\Gamma \cup \Delta$. Then there is some variable $x \in \mathcal{X}$ for which $v(x) = \mathbf{u}$ whereas $t(x) \neq \mathbf{u}$. The former implies $\langle v, t\rangle \not\models def(x)$ while the latter implies $def(x) \in \Delta$, so we conclude $\langle v, t\rangle \not\models \Delta$ reaching a contradiction.

Case 2. Suppose that $\langle t, t\rangle \models \Gamma'$. Take the theory $\Delta = \Delta_1 \cup \Delta_2$ with:

$$\Delta_1 = \{\, def(x) \mid x \in \mathcal{X},\ h(x) \neq \mathbf{u}\}$$

and Δ_2 consisting of all rules $def(x) \leftarrow def(y)$ for all pair variables x, y in the set:

$$\{z \in \mathcal{X} \mid h(z) = \mathbf{u}, t(z) \neq \mathbf{u}\}$$

We prove first that $\langle t, t\rangle$ is not an equilibrium model for $\Gamma \cup \Delta$ because $\langle h, t\rangle$ is, indeed, a model of this theory. To show this, it suffices to see that $\langle h, t\rangle \models \Delta$. First, it follows that $\langle h, t\rangle \models \Delta_1$ because Δ_1 only contains facts of the form $def(x)$ per each variable x satisfying $h(x) \neq \mathbf{u}$. Second, $\langle h, t\rangle \models \Delta_2$ also follows because for all the implications of the form of $def(x) \leftarrow def(y)$ in Δ_2, both $\langle h, t\rangle \not\models def(y)$ – because $h(y) = \mathbf{u}$ – and $\langle t, t\rangle \models def(x)$ – because $t(x) \neq \mathbf{u}$. It remains to be proven that $\langle t, t\rangle$ is an equilibrium model of $\Gamma' \cup \Delta$. We can see that $\langle t, t\rangle \models \Delta$ follows by persistence because $\langle h, t\rangle \models \Delta$, and thus, $\langle t, t\rangle \models \Gamma' \cup \Delta$. Suppose, for the sake of contradiction, that there existed some $v \subset t$ such that $\langle v, t\rangle \models \Gamma' \cup \Delta$. Since $\langle v, t\rangle \models \Delta_1$, any variable x defined in h must be defined in v as well, but as $v \subset t$, $v(x) = t(x) = h(x)$ and so, we conclude $h \subseteq v$. However, we also know $\langle h, t\rangle \not\models \Gamma' \subseteq \Gamma' \cup \Delta$, and so $v \neq h$, that is, $h \subset v \subset t$. Consider any variable y defined in v but not in h, that is, $h(y) = \mathbf{u}$ and $\mathbf{u} \neq v(y) = t(y)$ by persistence. Now, y is undefined in h and defined in t, and suppose we take any other variable x in the same situation. Then, we have an implication $def(x) \leftarrow def(y)$ in Δ_2 that must be satisfied by $\langle v, t\rangle$ whereas, on the other hand, $\langle v, t\rangle \models def(y)$ because $v(y) \neq \mathbf{u}$. As a result, $\langle v, t\rangle \models def(x)$ for all variables x undefined in h but not in t. But as $\langle v, t\rangle \models \Delta_1$ the same happens for variables defined in h. As a result, $\langle v, t\rangle \models def(x)$ iff $\langle t, t\rangle \models def(x)$ and this implies

https://doi.org/10.1017/S1471068426100647 Published online by Cambridge University Press

$v(x) = t(x)$ for all variables, namely, $v = t$ reaching a contradiction. Therefore, $\langle t, t\rangle$ is an equilibrium model of $\Gamma' \cup \Delta$. □

This proof is analogous to the original one for HT by Lifschitz *et al.* (2001), using here constraint atoms $def(x)$ to play the role of propositional atoms in the original proof.

4 Review: Logic programs with abstract theories as HT$_c$-theories

The main distinctive feature of `clingo 5` is the introduction of theory atoms in its syntax (Gebser *et al.* 2016). We review its most recent semantic characterization based on the concept of *abstract theories* (Cabalar et al. 2023). We consider an alphabet consisting of two disjoint sets, namely, a set $\mathcal{A}$ of *propositional* (or *regular*) *atoms* and a set $\mathcal{T}$ of *theory atoms*, whose truth is governed by some external theory. We use letters `a`, `s`, and b and variants of them for atoms in $\mathcal{A}$, $\mathcal{T}$, and $\mathcal{A} \cup \mathcal{T}$, respectively. In `clingo 5`, theory atoms are expressions preceded by '&', but their internal syntax is not predetermined: it can be defined by the user to build new extensions. As an example, the system `clingcon` extends the input language of `clingo` with linear equations, represented as theory atoms of the form

$$\&\texttt{sum}\{k_1 * x_1; \ldots; k_n * x_n\} \prec k_0 \tag{4}$$

where each x_i is an integer variable and each $k_i \in \mathbb{Z}$ an integer constant for $0 \leq i \leq n$, whereas $\prec$ is a comparison symbol such as `<=`, `=` , `!=`, `<`, `>`, `>=`. Several theory atoms may represent the same theory entity. For instance, &`sum`{x}>0 and &`sum`{`x`}>=1 actually represent the same condition (as linear equations).

A *literal* is any atom $b \in \mathcal{A} \cup \mathcal{T}$ or its default negation $\neg b$. A $\mathcal{T}$-program over $\langle \mathcal{A}, \mathcal{T}\rangle$ is a set of rules of the form

$$b_0 \leftarrow b_1, \ldots, b_n, \neg b_{n+1}, \ldots, \neg b_m \tag{5}$$

where $b_i \in \mathcal{A} \cup \mathcal{T}$ for $1 \leq i \leq m$ and $b_0 \in \mathcal{A} \cup \mathcal{T} \cup \{\bot\}$ with $\bot \notin \mathcal{A} \cup \mathcal{T}$ denoting the falsum constant. We sometimes identify (5) with the formula

$$b_1 \wedge \cdots \wedge b_n \wedge \neg b_{n+1} \wedge \cdots \wedge \neg b_m \rightarrow b_0.$$

We let notations $h(r) \stackrel{\text{def}}{=} b_0$, $B(r)^+ \stackrel{\text{def}}{=} \{b_1, \ldots, b_n\}$ and $B(r)^- \stackrel{\text{def}}{=} \{b_{n+1}, \ldots, b_m\}$ stand for the *head*, the *positive* and the *negative body atoms* of a rule r as in (5). The set of *body atoms* of r is just $B(r) \stackrel{\text{def}}{=} B(r)^+ \cup B(r)^-$. Finally, the sets of *body* and *head atoms* of a program P are defined as $B(P) \stackrel{\text{def}}{=} \bigcup_{r \in P} B(r)$ and $H(P) \stackrel{\text{def}}{=} \{h(r) \mid r \in P\} \setminus \{\bot\}$, respectively.

The semantics of $\mathcal{T}$-programs in `clingo` (Gebser *et al.* 2016), relies on a two-step process: (1) generate regular stable models (as in Definition 2) and (2) select the ones passing a theory certification. We present next this semantics following the formalization recently introduced by Cabalar *et al.* (2023).

An *abstract theory* $\mathfrak{T}$ is a triple $\langle \mathcal{T}, Sat, \widehat{\cdot}\rangle$ where $\mathcal{T}$ is the set of *theory atoms* constituting the language of the abstract theory. The set $Sat \subseteq 2^{\mathcal{T}}$ contains all possible sets of theory atoms that are considered satisfiable by the theory: each one of these sets $S \in Sat$

https://doi.org/10.1017/S1471068426100647 Published online by Cambridge University Press

is said to be $\mathfrak{T}$-*satisfiable*. Finally, $\widehat{\cdot}\colon \mathcal{T} \to \mathcal{T}$ is a function mapping theory atoms to their *complement* such that $\widehat{\widehat{\mathtt{s}}} = \mathtt{s}$ for any $\mathtt{s} \in \mathcal{T}$. We define $\widehat{S} = \{\widehat{\mathtt{s}} \mid \mathtt{s} \in S\}$ for any set $S \subseteq \mathcal{T}$.

We partition the set of theory atoms into two disjoint sets, namely, a set $\mathcal{E}$ of *external* theory atoms and a set $\mathcal{F}$ of *founded* theory atoms. Intuitively, the truth of each external atom in $\mathcal{E}$ requires no justification. Founded atoms on the other hand must be derived by the $\mathcal{T}$-program. We assume that founded atoms do not occur in the body of rules. We refer to the work by Janhunen *et al.* (2017) for a justification for this assumption.

Given any set S of theory atoms, we define its *(complemented) completion* with respect to external atoms $\mathcal{E}$, denoted by $Comp_{\mathcal{E}}(S)$, as:

$$Comp_{\mathcal{E}}(S) \stackrel{\text{def}}{=} S \ \cup \ \widehat{(\mathcal{E} \setminus S)}$$

In other words, we add the complement atom $\widehat{\mathtt{s}}$ for every external atom $\mathtt{s}$ that does not occur explicitly in S.

Definition 4 (Solution; Cabalar et al. 2023).
Given a theory $\mathfrak{T} = \langle \mathcal{T}, Sat, \widehat{\cdot} \rangle$ and a set $\mathcal{E} \subseteq \mathcal{T}$ of external theory atoms, a set $S \subseteq \mathcal{T}$ of theory atoms is a $\langle \mathfrak{T}, \mathcal{E} \rangle$-*solution*, if $S \in Sat$ and $Comp_{\mathcal{E}}(S) \in Sat$.

That is, S is a $\langle \mathfrak{T}, \mathcal{E} \rangle$-*solution* whenever both S and $Comp_{\mathcal{E}}(S)$ are $\mathfrak{T}$-satisfiable.

Definition 5 (Theory stable model[4]; Cabalar et al. 2023).
Given a theory $\mathfrak{T} = \langle \mathcal{T}, Sat, \widehat{\cdot} \rangle$ and a set $\mathcal{E} \subseteq \mathcal{T}$ of external theory atoms, a set $X \subseteq \mathcal{A} \cup \mathcal{T}$ of atoms is a $\langle \mathfrak{T}, \mathcal{E} \rangle$-*stable model* of a $\mathcal{T}$-program P, if there is some $\langle \mathfrak{T}, \mathcal{E} \rangle$-solution S such that X is a regular stable model of the program

$$P \cup \{\mathtt{s} \leftarrow \mid \mathtt{s} \in (S \cap \mathcal{E})\} \cup \{\bot \leftarrow \mathtt{s} \mid \mathtt{s} \in ((\mathcal{T} \cap H(P)) \setminus S)\} \ . \tag{6}$$

As an example of an abstract theory, consider the case of *linear equations* $\mathfrak{L}$ which can be used to capture `clingcon`-programs. This theory is formally defined as $\mathfrak{L} = \langle \mathcal{T}, Sat, \widehat{\cdot} \rangle$, where

- $\mathcal{T}$ is the set of all expressions of form (4),
- Sat is the set of all subsets $S \subseteq \mathcal{T}$ of expressions of form (4) for which there exists an assignment of integer values to their variables that satisfies all linear equations in S according to their usual meaning, and
- the complement function $\widehat{\&\mathtt{sum}\{\cdot\} \prec c}$ is defined as $\&\mathtt{sum}\{\cdot\} \mathbin{\widehat{\prec}} c$ with $\widehat{\prec}$ defined according to the following table:

$\prec$	<=	=	!=	<	>	>=
$\widehat{\prec}$	>	!=	=	>=	<=	<

[4] The only minor difference to the original definition of a $\langle \mathfrak{T}, \mathcal{E} \rangle$-stable model by Cabalar *et al.* (2023) is the characterization of regular stable models. While Cabalar *et al.* (2023) use the program reduct (Gelfond and Lifschitz 1988) we resort in Definition 2 to HT instead.

https://doi.org/10.1017/S1471068426100647 Published online by Cambridge University Press

Using the theory atoms of theory $\mathfrak{L}$, we can write the running example from the introduction as the $\mathcal{T}$-program:

$$\bot \leftarrow \neg \mathtt{a},\ \&\mathtt{sum}\{s\}>=120 \tag{7}$$

$$\mathtt{a} \leftarrow \&\mathtt{sum}\{s\}>100 \tag{8}$$

5 A new and simpler definition of theory stable model

In this section, we introduce a new, simplified version of the definition of $\langle \mathfrak{T}, \mathcal{E}\rangle$-*stable model* that, under certain conditions introduced below, is equivalent to Definition 5. This streamlined definition not only enhances the clarity of the technical development presented in the subsequent sections but also holds the potential to facilitate future advancements in hybrid solver design.

Definition 6 (Theory stable model simplified).
Given a theory $\mathfrak{T} = \langle \mathcal{T}, Sat, \widehat{\cdot}\rangle$ and a set $\mathcal{E} \subseteq \mathcal{T}$ of external theory atoms, a set $X \subseteq \mathcal{A} \cup \mathcal{T}$ of atoms is a $\langle \mathfrak{T}, \mathcal{E}\rangle$-*stable model* of a $\mathcal{T}$-program P, if $(X \cap \mathcal{T}) \in Sat$ and X is a regular stable model of the theory

$$P \cup \{\mathtt{s} \vee \widehat{\mathtt{s}} \mid \mathtt{s} \in \mathcal{E}\}. \tag{9}$$

This definition drops the existential quantifier used in Definition 5 for identifying a $\langle \mathfrak{T}, \mathcal{E}\rangle$-solution S.

To state the conditions under which Definitions 5 and 6 are equivalent, we need the following concepts (Cabalar *et al.* 2023). An abstract theory $\mathfrak{T} = \langle \mathcal{T}, Sat, \widehat{\cdot}\rangle$ is *consistent* if none of its satisfiable sets contains complementary theory atoms, that is, there is no $S \in Sat$ such that $\mathtt{s} \in S$ and $\widehat{\mathtt{s}} \in S$ for some atom $\mathtt{s} \in \mathcal{T}$. A set S of theory atoms is *closed* if $\mathtt{s} \in S$ implies $\widehat{\mathtt{s}} \in S$. A set S of theory atoms is called $\mathcal{E}$-*complete*, if for all $\mathtt{s} \in \mathcal{E}$, either $\mathtt{s} \in S$ or $\widehat{\mathtt{s}} \in S$. A theory $\mathfrak{T} = \langle \mathcal{T}, Sat, \widehat{\cdot}\rangle$ is *monotonic* if $S \subseteq S'$ and $S' \in Sat$ implies $S \in Sat$. Programs over a consistent theory with a closed set of external theory atoms have the following interesting properties:

Proposition 3 (Cabalar et al. 2023, Proposition 2).
For a consistent abstract theory $\mathfrak{T} = \langle \mathcal{T}, Sat, \widehat{\cdot}\rangle$ and a closed set $\mathcal{E} \subseteq \mathcal{T}$ of external theory atoms, all $\langle \mathfrak{T}, \mathcal{E}\rangle$-solution s $S \subseteq \mathcal{T}$ are $\mathcal{E}$-complete.

Theorem 3.
Given a theory $\mathfrak{T} = \langle \mathcal{T}, Sat, \widehat{\cdot}\rangle$ and a set $\mathcal{E} \subseteq \mathcal{T}$ of external theory atoms, if $\mathfrak{T}$ is consistent and monotonic, and $\mathcal{E}$ is closed, then Definitions 5 and 6 are equivalent.

The preconditions of Theorem 3 cover many hybrid extensions of `clingo` such as `clingcon`, `clingo`[DL], and `clingo`[LP]. Therefore, in the rest of the paper, we assume these conditions hold and use Definition 6 as the definition of a $\langle \mathfrak{T}, \mathcal{E}\rangle$-stable model.

6 Review: Structured and compositional theories

The approach presented in Sections 4 and 5 is intentionally generic in its formal definitions. No assumption is made on the syntax or inner structure of theory atoms. An

https://doi.org/10.1017/S1471068426100647 Published online by Cambridge University Press

abstract theory is only required to specify when a set of theory atoms is satisfiable and provide a complement function for each theory atom. Definition 6 is a bit more specific, and further assumes consistent and monotonic theories (with a closed set of external atoms), something we may expect in most cases while still being very generic. The advantage of this generality is that it allows us to accommodate external theories without requiring much knowledge about their behavior. However, this generality comes at a price: ignoring the structure of the external theory may prevent in depth formal elaborations, such as, for instance, the study of strong equivalence for logic programs with constraint atoms.

Also, in many practical applications of hybrid systems, we are interested in the assignment of values to variables rather than the theory atoms that are satisfied, something not reflected in Definitions 5 or 6. This is what happens, for instance, in most hybrid extensions of answer set solvers including the hybrid versions of `clingo`, namely, `clingcon`, `clingo`[DL], and `clingo`[LP]. Since the presence of variables in theory atoms can be exploited to describe their semantics in more detail, we resort to refined types of theories that are enriched with a specific structure.

An *assignment* over $\mathcal{X}, \mathcal{D}$ is a valuation over $\mathcal{X}, \mathcal{D}$ such that $v(x) \in \mathcal{D}$ for all $x \in \mathcal{X}$.

Definition 7 (Structured theory; Cabalar et al. 2023).
Given an abstract theory $\mathfrak{T} = \langle \mathcal{T}, Sat, \widehat{\cdot} \rangle$, we define its *structure* as a tuple $(\mathcal{X}_\mathfrak{T}, \mathcal{D}_\mathfrak{T}, vars_\mathfrak{T}, [\![\cdot]\!]_\mathfrak{T})$, where

1. $\mathcal{X}_\mathfrak{T}$ is a set of variables,
2. $\mathcal{D}_\mathfrak{T}$ is a set of domain elements,
3. $vars_\mathfrak{T} : \mathcal{T} \to 2^{\mathcal{X}_\mathfrak{T}}$ is a function returning the set of variables contained in a theory atom such that $vars_\mathfrak{T}(\mathtt{s}) = vars_\mathfrak{T}(\widehat{\mathtt{s}})$ for all theory atoms $\mathtt{s} \in \mathcal{T}$,
4. $[\![\cdot]\!]_\mathfrak{T} : \mathcal{T} \to 2^{\mathcal{V}_\mathfrak{T}}$ is a function mapping theory atoms to sets of assignments, where $\mathcal{V}_\mathfrak{T}$ denotes the set of all assignments over $\mathcal{X}_\mathfrak{T}$ and $\mathcal{D}_\mathfrak{T}$, and satisfying

$$v \in [\![\mathtt{s}]\!]_\mathfrak{T} \text{ iff } w \in [\![\mathtt{s}]\!]_\mathfrak{T}$$

for all theory atoms $\mathtt{s} \in \mathcal{T}$ and every pair of valuations v, w agreeing on the value of all variables $vars_\mathfrak{T}(\mathtt{s})$ occurring in $\mathtt{s}$.

Whenever an abstract theory $\mathfrak{T}$ is associated with such a structure, we call it *structured* (rather than abstract). Observe that structured theories are based on the same concepts as HT_c, viz. a domain, a set of variables, valuation functions, and denotations for atoms. In fact, we assume that the previously introduced definitions and notation for these concepts still apply here. This similarity is intended, since HT_c was originally thought of as a generalization of logic programs with theory atoms. One important difference between an HT_c valuation and a structured theory valuation $v \in \mathcal{V}_\mathfrak{T}$ is that the latter cannot leave a variable undefined in HT_c terms, that is, for a structured theory, we assume that the "undefined" value is not included in the domain $\mathbf{u} \notin \mathcal{D}_\mathfrak{T}$, and so, $v(x) \neq \mathbf{u}$ for all $x \in \mathcal{X}_\mathfrak{T}$ and $v \in \mathcal{V}_\mathfrak{T}$. We also emphasize the use of the theory name $\mathfrak{T}$ subindex in all the components of a structure because, as we see below, an HT_c characterization allows us to accommodate multiple theories in the same formalization.

https://doi.org/10.1017/S1471068426100647 Published online by Cambridge University Press

Given a set S of theory atoms, we define its denotation as $[\![S]\!]_{\mathfrak{T}} \stackrel{\text{def}}{=} \bigcap_{\mathtt{s} \in S} [\![\mathtt{s}]\!]_{\mathfrak{T}}$. For any structured theory $\mathfrak{T} = \langle \mathcal{T}, Sat, \widehat{\cdot} \rangle$ with structure $(\mathcal{X}_{\mathfrak{T}}, \mathcal{D}_{\mathfrak{T}}, vars_{\mathfrak{T}}, [\![\cdot]\!]_{\mathfrak{T}})$, if $X \subseteq \mathcal{A} \cup \mathcal{T}$ is a set of atoms, we define $ans(X)$ as the set

$$\{(Y, v|_{\Sigma}) \mid v \in [\![X \cap \mathcal{T}]\!]_{\mathfrak{T}}\} \quad \text{with } Y = X \cap \mathcal{A} \text{ and } \Sigma = vars_{\mathfrak{T}}(X \cap \mathcal{T}).$$

Intuitively, $ans(X)$ collects all pairs (Y, v') where Y is fixed to the propositional atoms in X and v' varies among all valuations in $[\![X \cap \mathcal{T}]\!]_{\mathfrak{T}}$ restricted to the variables occurring in the theory atoms in X. Note that v is only defined for a subset of variables, namely, $\Sigma = vars_{\mathfrak{T}}(X \cap \mathcal{T}) \subseteq \mathcal{X}$.

Definition 8 (Answer set; Cabalar et al. 2023).
If X is a $\langle \mathfrak{T}, \mathcal{E} \rangle$-stable model of program P and $(Y, v) \in ans(X)$, then (Y, v) is a $\langle \mathfrak{T}, \mathcal{E} \rangle$-*answer set* of P.

Let $\mathfrak{T} = \langle \mathcal{T}, Sat, \widehat{\cdot} \rangle$ be a theory with structure $(\mathcal{X}_{\mathfrak{T}}, \mathcal{D}_{\mathfrak{T}}, vars_{\mathfrak{T}}, [\![\cdot]\!]_{\mathfrak{T}})$. We say that $\mathfrak{T}$ has an *absolute complement* whenever the denotation of $\widehat{\mathtt{s}}$ is precisely the set complement of $[\![\mathtt{s}]\!]_{\mathfrak{T}}$, that is, $[\![\widehat{\mathtt{s}}]\!]_{\mathfrak{T}} = \mathcal{V}_{\mathfrak{T}} \setminus [\![\mathtt{s}]\!]_{\mathfrak{T}}$. We also say that $\mathfrak{T}$ is *compositional*, if it satisfies $Sat = \{S \subseteq \mathcal{T} \mid [\![S]\!]_{\mathfrak{T}} \neq \emptyset\}$, that is, a set S is $\mathfrak{T}$-satisfiable iff its denotation is not empty. This means that, for compositional theories, the set Sat of $\mathfrak{T}$-satisfiable sets does not need to be explicitly stated as it can be derived from the denotation. Hence, we can write $\mathfrak{T} = \langle \mathcal{T}, \widehat{\cdot}, \mathcal{X}_{\mathfrak{T}}, \mathcal{D}_{\mathfrak{T}}, vars_{\mathfrak{T}}, [\![\cdot]\!]_{\mathfrak{T}} \rangle$ to denote a compositional theory.

As an example for a compositional, structured theory with an absolute complement, let us associate the theory of linear equations $\mathfrak{L}$ with the structure $(\mathcal{X}_{\mathfrak{L}}, \mathcal{D}_{\mathfrak{L}}, vars_{\mathfrak{L}}, [\![\cdot]\!]_{\mathfrak{L}})$, where

- $\mathcal{X}_{\mathfrak{L}}$ is an infinite set of integer variables,
- $\mathcal{D}_{\mathfrak{L}} = \mathbb{Z}$,
- $vars_{\mathfrak{L}}(\&\mathtt{sum}\{k_1 * x_1; \ldots; k_n * x_n\} \prec k_0) = \{x_1, \ldots, x_n\}$, and
- $[\![\&\mathtt{sum}\{k_1 * x_1; \ldots; k_n * x_n\} \prec k_0]\!]_{\mathfrak{L}} =$

$$\{v \in \mathcal{V}_{\mathfrak{L}} \mid \{k_1, v(x_1), \ldots k_n, v(x_n)\} \subseteq \mathbb{Z} \text{ and } \sum\nolimits_{1 \leq i \leq n} k_i * v(x_i) \prec k_0\} \ .$$

With $\mathfrak{L}$, a set S of theory atoms capturing linear equations is $\mathfrak{L}$-satisfiable whenever $[\![S]\!]_{\mathfrak{L}}$ is non-empty.

In general, we have a one-to-many correspondence between $\langle \mathfrak{T}, \mathcal{E} \rangle$-stable models and their associated $\langle \mathfrak{T}, \mathcal{E} \rangle$-answer sets. That is, the cardinality of $ans(X)$ is generally greater than one for a single stable model X. However, if we focus on consistent, compositional theories that have an absolute complement, then we can establish a one-to-one correspondence between any $\langle \mathfrak{T}, \mathcal{E} \rangle$-stable model of a program and a kind of equivalence classes among its associated $\langle \mathfrak{T}, \mathcal{E} \rangle$-answer sets. Each of the equivalence classes may contain many $\langle \mathfrak{T}, \mathcal{E} \rangle$-answer sets, but any of them has enough information to reconstruct the corresponding $\langle \mathfrak{T}, \mathcal{E} \rangle$-stable model. An answer set (Y, v) *satisfies* an atom $b \in \mathcal{A} \cup \mathcal{T}$, written $(Y, v) \models b$, if

- $b \in Y$ for propositional atoms $b \in \mathcal{A}$ and
- $v \hat{\in} [\![b]\!]_{\mathfrak{T}}$ for theory atoms $b \in \mathcal{T}$,

where $v \hat{\in} [\![b]\!]_{\mathfrak{T}}$ holds iff there exists some valuation $w \in [\![b]\!]_{\mathfrak{T}}$ such that v and w agree on the variables in $dom(v)$. Note that we use $v \hat{\in} [\![b]\!]_{\mathfrak{T}}$ instead of $v \in [\![b]\!]_{\mathfrak{T}}$ because v

https://doi.org/10.1017/S1471068426100647 Published online by Cambridge University Press

can be partial and, if so, we just require that there exists some complete valuation in $[\![b]\!]_{\mathfrak{T}}$ that agrees with the values assigned by v to its defined variables $dom(v)$. For any negative literal $\neg b$, we say that $(Y, v) \models \neg b$ simply when $(Y, v) \not\models b$. If B is a rule body, we write $(Y, v) \models B$ to stand for $(Y, v) \models \ell$ for every literal ℓ in B. For a program P and an $\langle \mathfrak{T}, \mathcal{E} \rangle$-*answer set* (Y, v), we define

$$stb_P(Y, v) = Y \cup \{\mathtt{s} \in \mathcal{E} \mid v \,\hat{\in}\, [\![\mathtt{s}]\!]_{\mathfrak{T}}\} \cup \{\mathtt{s} \in \mathcal{F} \mid (\mathtt{s} \leftarrow B) \in P, (Y, v) \models B\}$$

We also write $(Y_1, v_1) \sim (Y_2, v_2)$ if $stb_P(Y_1, v_1) = stb_P(Y_2, v_2)$ and say that (Y_1, v_1) and (Y_2, v_2) belong to the same equivalence class with respect to P.

Proposition 4 (Cabalar et al. 2023, Proposition 8).
Let $\mathfrak{T} = \langle \mathcal{T}, \hat{\cdot}, \mathcal{X}_{\mathfrak{T}}, \mathcal{D}_{\mathfrak{T}}, vars_{\mathfrak{T}}, [\![\cdot]\!]_{\mathfrak{T}} \rangle$ be a compositional theory with an absolute complement and let $\mathcal{E} \subseteq \mathcal{T}$ be a closed set of external atoms. There is a one-to-one correspondence between the $\langle \mathfrak{T}, \mathcal{E} \rangle$-stable models of a program P and the equivalence classes with respect to P of its $\langle \mathfrak{T}, \mathcal{E} \rangle$-answer sets.

Furthermore, if (Y, v) is a $\langle \mathfrak{T}, \mathcal{E} \rangle$-answer set, then $stb_P(Y, v)$ is a $\langle \mathfrak{T}, \mathcal{E} \rangle$-stable model of P and (Y, v) belongs to $ans(stb_P(Y, v))$.

7 HT$_c$-characterization of logic programs with structured and compositional theories

We now present a direct encoding of $\mathcal{T}$-programs as HT$_c$ theories. This encoding is "direct" in the sense that it preserves the structure of each program rule by rule and atom by atom, only requiring the addition of a fixed set of axioms.

In the following, we restrict ourselves to consistent compositional theories with an absolute complement and refer to them just as theories.

We start embodying compositional theories in HT$_c$ by mapping their respective structures (domain, variables, valuations, and denotations), while having in mind that HT$_c$ may tolerate multiple abstract theories in the same formalization. For this reason, when we encode a theory $\mathfrak{T} = \langle \mathcal{T}_{\mathfrak{T}}, \hat{\cdot}, \mathcal{X}_{\mathfrak{T}}, \mathcal{D}_{\mathfrak{T}}, vars_{\mathfrak{T}}, [\![\cdot]\!]_{\mathfrak{T}} \rangle$ into an HT$_c$ theory over a signature $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, we only require $\mathcal{X}_{\mathfrak{T}} \subseteq \mathcal{X}$ and $\mathcal{D}_{\mathfrak{T}} \subseteq \mathcal{D}$, so that the HT$_c$ signature may also include variables and domain values other than the ones in $\mathfrak{T}$. We then map each abstract theory atom $\mathtt{s} \in \mathcal{T}$ into a corresponding HT$_c$-constraint atom $\tau(\mathtt{s}) \in \mathcal{C}$ with the same variables $vars(\tau(\mathtt{s})) = vars_{\mathfrak{T}}(\mathtt{s})$. We also require the following relation between the theory denotation of a theory atom and the HT$_c$ denotation of its corresponding constraint atom:

$$[\![\tau(\mathtt{s})]\!] \stackrel{\text{def}}{=} \{v \in \mathcal{V}^{\mathcal{X},\mathcal{D}} \mid \exists w \in [\![\mathtt{s}]\!]_{\mathfrak{T}},\ v|_{\mathcal{X}_{\mathfrak{T}}} = w\} \tag{10}$$

Note that HT$_c$ valuations $v \in \mathcal{V}^{\mathcal{X},\mathcal{D}}$ apply to a (possibly) larger set of variables $\mathcal{X} \supseteq \mathcal{X}_{\mathfrak{T}}$ and on larger domains $\mathcal{D}_{\mathbf{u}} \supset \mathcal{D}_{\mathfrak{T}}$, which include the element $\mathbf{u} \notin \mathcal{D}_{\mathfrak{T}}$ to represent undefined variables in HT$_c$. The denotation $[\![\tau(\mathtt{s})]\!]$ collects all possible HT$_c$-valuations that coincide with some $\mathfrak{T}$-valuation $w \in [\![\mathtt{s}]\!]_{\mathfrak{T}}$ on the variables of theory $\mathfrak{T}$, letting everything else vary vary freely.

We write $\tau(S)$ for $\{\tau(\mathtt{s}) \mid \mathtt{s} \in S\}$ and $S \subseteq \mathcal{T}$. The following result shows that this mapping of denotations preserves $\mathfrak{T}$-satisfiability:

https://doi.org/10.1017/S1471068426100647 Published online by Cambridge University Press

Proposition 5.
Given a theory $\mathfrak{T} = \langle \mathcal{T}_{\mathfrak{T}}, \widehat{\ }, \mathcal{X}_{\mathfrak{T}}, \mathcal{D}_{\mathfrak{T}}, vars_{\mathfrak{T}}, [\![\cdot]\!]_{\mathfrak{T}} \rangle$, *a set* $S \subseteq \mathcal{T}$ *of theory atoms is* $\mathfrak{T}$*-satisfiable iff* $\tau(S)$ *is satisfiable in* HT_c.

Let us now consider $\mathcal{T}$-programs P over $\langle \mathcal{A}, \mathcal{T} \rangle$, that are associated with a structured theory $\mathfrak{T} = \langle \mathcal{T}_{\mathfrak{T}}, \widehat{\ }, \mathcal{X}_{\mathfrak{T}}, \mathcal{D}_{\mathfrak{T}}, vars_{\mathfrak{T}}, [\![\cdot]\!]_{\mathfrak{T}} \rangle$. As explained in Section 2, we can capture regular programs by identifying propositional atoms $\mathcal{A}$ with both variables and constraint atoms and having the truth constant $\mathbf{t}$ in the domain. We can capture this by adding propositional atoms both as variables and as constraint atoms to the signature of the HT_c theory, so we have

$$\mathcal{A} \cup \mathcal{X}_{\mathfrak{T}} \subseteq \mathcal{X} \tag{11}$$

$$\mathcal{A} \cup \{\tau(\mathtt{s}) \mid \mathtt{s} \in \mathcal{T}_{\mathfrak{T}}\} \subseteq \mathcal{C} \tag{12}$$

and the truth constant $\mathbf{t}$ in the domain, so we have $\mathcal{D}_{\mathfrak{T}} \cup \{\mathbf{t}\} \subseteq \mathcal{D}$. For each propositional atom $\mathtt{a} \in \mathcal{A}$, its variables and denotation are defined as follows:

- $vars(\mathtt{a}) = \{a\}$, and
- $[\![\mathtt{a}]\!] = [\![a : \{\mathbf{t}\}]\!] = \{v \in \mathcal{V}^{\mathcal{X},\mathcal{D}} \mid v(a) = \mathbf{t}\}$.

With this encoding, each propositional atom $\mathtt{a} \in \mathcal{A}$ is mapped into itself in the HT_c-theory, viz. $\tau(\mathtt{a}) \stackrel{\text{def}}{=} \mathtt{a}$.

Since the resulting HT_c-theory contains variables from the abstract theory and the propositional atoms, we are interested in forcing variables to range only over their corresponding subdomain, one from the external theory and one for Boolean values. This is achieved by including an axiom of form

$$def(x) \rightarrow x : \mathcal{D}_{\mathfrak{T}} \tag{13}$$

for each $x \in \mathcal{X}_{\mathfrak{T}}$ and each abstract theory $\mathfrak{T}$ encoded in our HT_c formalization. The application of any valuation $v \in \mathcal{V}^{\mathcal{X}_{\mathfrak{T}},\mathcal{D}}$ satisfying (13) to a variable $x \in \mathcal{X}_{\mathfrak{T}}$ from the abstract theory returns some element from the theory domain $v(x) \in \mathcal{D}_{\mathfrak{T}}$ or is undefined, viz. $v(x) = \mathbf{u}$. Let us denote by $\delta(\mathfrak{T}, \mathcal{A})$ the set of all the axioms of the form of (13) for each $x \in \mathcal{X}_{\mathfrak{T}}$ plus all the axioms of the form (1) for every $\mathtt{a} \in \mathcal{A}$.

We are now ready to introduce the direct translation of a $\mathcal{T}$-program P over $\langle \mathcal{A}, \mathcal{T} \rangle$ with external theory $\mathfrak{T}$ into an HT_c-theory $\tau(P, \mathfrak{T}, \mathcal{E})$ where $\mathcal{E}$ is a set of theory atoms considered external in P. Given any rule r of the form of (5), we let $\tau^B(r)$ stand for the formula

$$\tau(b_1) \wedge \cdots \wedge \tau(b_n) \wedge \neg\tau(b_{n+1}) \wedge \cdots \wedge \neg\tau(b_m) \tag{14}$$

representing the body of r, where each occurrence of an atom b_i in the program is replaced by $\tau(b_i)$. We extend the application of τ to the whole rule r and let $\tau(r)$ stand for

$$\tau^B(r) \rightarrow \tau(b_0) \tag{15}$$

assuming that, when the head is $b_0 = \bot$, its translation is simply $\tau(\bot) \stackrel{\text{def}}{=} \bot$. We further write $\tau(P) \stackrel{\text{def}}{=} \{\tau(r) \mid r \in P\}$, applying our transformation to all rules in program P. The

https://doi.org/10.1017/S1471068426100647 Published online by Cambridge University Press

complete translation of $\mathcal{T}$-program P is defined as

$$\tau(P, \mathfrak{T}, \mathcal{E}) \stackrel{\text{def}}{=} \tau(P) \cup \sigma(\mathfrak{T}, \mathcal{E}) \cup \delta(\mathfrak{T}, \mathcal{A}) \tag{16}$$

where $\sigma(\mathfrak{T}, \mathcal{E})$ consists of

$$def(x) \quad \text{for every } x \in vars(\mathtt{s}) \text{ and } \mathtt{s} \in \mathcal{E}. \tag{17}$$

Formula (17) asserts that each variable in any external atom can be arbitrarily assigned some (defined) value.

Proposition 6.
Let P be a $\mathcal{T}$-program over $\langle \mathcal{A}, \mathcal{T} \rangle$ with external theory $\mathfrak{T}$ and external atoms $\mathcal{E}$. For every $\mathtt{s} \in \mathcal{E}$,

$$\tau(P, \mathfrak{T}, \mathcal{E}) \models \tau(\mathtt{s}) \vee \tau(\widehat{\mathtt{s}}).$$

This proposition means that either $\mathtt{s}$ or its complement $\widehat{\mathtt{s}}$ must hold. In other words, axioms of form (17) entail a kind of *strong excluded middle* for external atoms similarly to the simplified definition of a $\langle \mathfrak{T}, \mathcal{E} \rangle$-stable model (in Definition 6). This is a stronger version of the usual choice construct $\tau(\mathtt{s}) \vee \neg\tau(\mathtt{s})$ in HT: we can freely add $\tau(\mathtt{s})$ or not but, when the latter happens, we further provide evidence for the complement $\tau(\widehat{\mathtt{s}})$.

Following our running example, consider program P_0 consisting of rules (7) and (8), plus the fact

$$\&\mathtt{sum}\{s\} = 130. \tag{18}$$

This program has a unique answer set where $\mathtt{a}$ is true and s is assigned the value 130.

We assume the following translation of constraint atoms:

$$\begin{aligned} \tau(\&\mathtt{sum}\{s\} >= 120) &\stackrel{\text{def}}{=} s \geq 120 \\ \tau(\&\mathtt{sum}\{s\} > 100) &\stackrel{\text{def}}{=} s > 100 \\ \tau(\&\mathtt{sum}\{s\} = 130) &\stackrel{\text{def}}{=} s = 130 \end{aligned}$$

Hence, $\tau(P_0)$ is the HT_c theory $\{(2), (3), s = 130\}$. Given that constraint atoms $\&\mathtt{sum}\{s\} \geq 120$ and $\&\mathtt{sum}\{s\} > 100$ occur in the body, they must be external. We assume that constraint atom (18) is not external (see discussion below). Thus, $\tau(P_0, \mathfrak{L}, \mathcal{E})$ is the result of adding to $\tau(P_0)$ the following axioms:

$$def(a) \rightarrow \mathtt{a} \tag{1}$$

$$def(s) \rightarrow s : \mathcal{D}_{\mathfrak{L}} \tag{19}$$

$$def(s) \tag{20}$$

We can replace the last two axioms simply by $s : \mathcal{D}_{\mathfrak{L}}$, which ensures that the variable s is assigned a value from the domain $\mathcal{D}_{\mathfrak{L}}$ of linear constraints, namely, an integer. Recall that (1) ensures that a is assigned a Boolean value, either $\mathtt{t}$or $\mathbf{u}$. This HT_c-theory has a unique stable model

$$\{a \mapsto \mathbf{t},\ s \mapsto 130\}.$$

https://doi.org/10.1017/S1471068426100647 Published online by Cambridge University Press

This stable model corresponds to the unique answer set of program P_0.

The following theorem states the relation between the answer sets of a $\mathcal{T}$-program P and the equilibrium models of its translation $\tau(P, \mathfrak{T}, \mathcal{E})$, generalizing the correspondence illustrated in the previous example.

Theorem 4.
Let P be a $\mathcal{T}$-program over $\langle \mathcal{A}, \mathcal{T} \rangle$ with $\mathcal{E}$ being a closed set of external atoms, and let $\mathfrak{T} = \langle \mathcal{T}, \mathcal{S}, \widehat{\cdot} \rangle$ be a consistent, compositional theory. Then, there is a one-to-one correspondence between the $\langle \mathfrak{T}, \mathcal{E} \rangle$-answer sets of P and the equilibrium models of $\tau(P, \mathfrak{T}, \mathcal{E})$ such that (Y, v) is a $\langle \mathfrak{T}, \mathcal{E} \rangle$-answer set of P iff $\langle t, t \rangle$ is an equilibrium model of theory $\tau(P, \mathfrak{T}, \mathcal{E})$ with $t = v \cup \{p \mapsto \mathbf{t} \mid p \in Y\}$.

This result shows that the semantics of a $\mathcal{T}$-program P can alternatively be described as the equilibrium models of the HT_c-theory $\tau(P, \mathfrak{T}, \mathcal{E})$. Intuitively, formulas of form (15) capture the rules in the $\mathcal{T}$-program and are used for the same purpose, that is, to decide which founded atoms from $\mathcal{T} \setminus \mathcal{E}$ can be eventually derived. Furthermore, due to the minimization imposed to obtain an equilibrium model $\langle t, t \rangle$, if a founded atom $\mathtt{s}$ is not derived (i.e., $\langle t, t \rangle \not\models \tau(\mathtt{s})$), then all its variables $x \in \mathit{vars}_{\mathfrak{T}}(\mathtt{s})$ not occurring in external atoms or other derived atoms are left undefined, viz. $t(x) = \mathbf{u}$.

An interesting consequence of the characterization provided by Theorem 4 is that all atoms whose variables occur in external atoms are implicitly external, even if we do not declare them as that. This result is trivial when looked at the definition of $\tau(P, \mathfrak{T}, \mathcal{E})$, but it is far from obvious when considering the definition of $\langle \mathfrak{T}, \mathcal{E} \rangle$-answer sets based on Definitions 5 or 6.

Corollary 1.
Let P be $\mathcal{T}$-programs over $\langle \mathcal{A}, \mathcal{T} \rangle$ where $\mathcal{E}$ and $\mathcal{E}'$ are sets of external theory atoms such that $\mathcal{E} \subseteq \mathcal{E}'$, and let $\mathfrak{T}$ be a consistent, compositional theory. If every $\mathtt{s} \in \mathcal{E}'$ satisfies $\mathit{vars}(\mathtt{s}) \subseteq \mathit{vars}(\mathcal{E})$, then the $\langle \mathfrak{T}, \mathcal{E} \rangle$-answer sets of P and the $\langle \mathfrak{T}, \mathcal{E}' \rangle$-answer sets of P coincide.

In other words, extending the set of external atoms by adding new ones whose variables are already occurring in other external atoms does not change the answer sets of the program. For instance, continuing with our running example, we have assumed that constraint atom (18) is not external, but because its only variable s occurs in external atoms, it is implicitly external. As stated above, assuming that it is external instead, does not change the answer sets of the program.

8 Strong equivalence of $\mathcal{T}$-programs

So far, the notion of strong equivalence applies to HT_c-theories, but not to $\mathcal{T}$-programs. We now leverage the translation introduced in Section 7 to lift strong equivalence to $\mathcal{T}$-programs.

Definition 9 (Strong equivalence of $\mathcal{T}$-programs).
$\mathcal{T}$-programs P and Q are *strongly equivalent* with respect to an external theory $\mathfrak{T}$ and a set $\mathcal{E}$ of theory atoms considered external, whenever HT_c-theories $\tau(P, \mathfrak{T}, \mathcal{E})$ and $\tau(Q, \mathfrak{T}, \mathcal{E})$ are strongly equivalent.

https://doi.org/10.1017/S1471068426100647 Published online by Cambridge University Press

Table 1. *Summary of complexity results*

	Unrestricted	Oracle	Polynomial
Satisfiability	Undecidable	NP^{O}	NP
Strong equivalence	Undecidable	$\mathsf{coNP}^{\mathsf{O}}$	coNP

This table summarizes the complexity results for $\mathcal{T}$-programs. The first column considers unrestricted $\mathcal{T}$-programs, the second column considers $\mathcal{T}$-programs where the complexity of the satisfiability problem of the external theory is in some complexity class O, and the third column considers $\mathcal{T}$-programs where the satisfiability problem of the external theory is decidable in polynomial time.

We can easily observe that, if $\mathcal{T}$-programs P and Q are strongly equivalent, then $P \cup R$ and $Q \cup R$ have the same answer sets for any program R. Our main result is an immediate consequence of Theorems 1 and 2 and lifts the characterization of strong equivalence from HT_c to $\mathcal{T}$-programs.

Theorem 5 (Main Result).
Two $\mathcal{T}$-programs are strongly equivalent with respect to an external theory and a set of theory atoms considered external iff their translations are equivalent in HT_c.

We now revisit the question posed in the introduction: can we remove rule (7) from any program containing rules (7) and (8) without changing its semantics? Continuing with our running example, recall that

$$\tau(P_1, \mathfrak{L}, \mathcal{E}) \;=\; \{(1), (2), (3), (19), (20)\} \tag{21}$$

where $P_1 = \{(7), (8)\}$. We saw in Section 3 that $\{(2), (3)\}$ and (3) are equivalent in HT_c. Therefore, (21) is equivalent in HT_c to

$$\tau(P_2, \mathfrak{L}, \mathcal{E}) = \{(1), (3), (19), (20)\}$$

where $P_2 = \{(8)\}$. By Theorem 5, this means that programs P_1 and P_2 are strongly equivalent, and, consequently, rule (7) can be safely removed from any program containing both rules without changing its semantics.

9 Complexity

In this section, we address the computational complexity of the satisfiability and strong equivalence problems for $\mathcal{T}$-programs. Table 1 summarizes our results, which we discuss in detail below. The complexity of $\mathcal{T}$-programs highly depends on the associated external theory $\mathfrak{T}$, and is in general undecidable as illustrated by the following example.

Example 1.
Let $\mathfrak{DE} = \langle \mathcal{T}, \frown, \mathcal{X}_{\mathfrak{T}}, \mathcal{D}_{\mathfrak{T}}, vars_{\mathfrak{T}}, [\![\cdot]\!]_{\mathfrak{T}} \rangle$ be a structured theory with variables $\mathcal{X}_{\mathfrak{T}} = \{x_1, x_2, \dots\}$, whose domain is the integers and whose atoms are of the form

$$\&\mathtt{sum}\{c_1 \cdot x_1^{e_1}, \dots, c_n \cdot x_n^{e_n}\} = 0$$

https://doi.org/10.1017/S1471068426100647 Published online by Cambridge University Press

where each $c_i \in \mathbb{Z}$, $e_i \in \mathbb{N}$ and x_i is a variable. The denotation $[\![\cdot]\!]_{\mathfrak{T}}$ of these atoms is the set of all assignments of integer values to the variables that satisfy the corresponding Diophantine equation.

Since the satisfiability problem for Diophantine equations is undecidable (Matiyasevich 1972), the satisfiability problem for $\mathcal{T}$-programs with theory $\mathfrak{DE}$ is also undecidable. The *satisfiability problem* for $\mathcal{T}$-program s consists of deciding whether there is a set of atoms that is a $\langle \mathfrak{T}, \mathcal{E} \rangle$-stable model of some program.

Theorem 6.
The satisfiability problem for $\mathcal{T}$-programs with theory $\mathfrak{DE}$ is undecidable, even if the program is restricted to be a single fact.

We can provide a more detailed analysis of the complexity in terms of oracles that can solve the satisfiability problem for the external theory $\mathfrak{T}$. In what follows, we assume that the reader is familiar with the basic concepts of complexity theory.[5] For convenience, we briefly recapitulate the definitions and some elementary properties of the complexity classes considered in our analysis. The class NP consists of all decision problems that can be solved by a non-deterministic Turing machine in polynomial time. As usual, for any complexity class C, by $\mathsf{co\text{-}C}$ we understand the class of all problems which are complementary to the problems in C. Thus, $\mathsf{co\text{-}NP}$ is the class of all problems whose complements are in NP. A non-deterministic *Turing machine with an oracle* is a non-deterministic Turing machine that has a special *query tape* and three special states: *query*, *yes*, and *no*. The computations of a Turing machine with oracle O proceed as usual, except when the machine enters the query state. In such case, the next state is either the yes state or the no state, depending on whether the string written on the query tape is in the language decided by the oracle O. For complexity classes C and O, we denote by C^{O} the class of languages which can be decided by a Turing machine with oracle O of the same sort and time bound as C (Papadimitriou 1994, pp. 339–340). For example, NP^{O} is the class of all decision problems that can be solved by a non-deterministic Turing machine with oracle O in polynomial time.

Theorem 7.
The satisfiability problem for $\mathcal{T}$-program s with a theory $\mathfrak{T}$ is decidable in NP^{O} where O is an oracle that solves the satisfiability problem for the external theory $\mathfrak{T}$.

Since regular ASP programs are a particular case of $\mathcal{T}$-programs, the following result immediately follows from the NP-completeness of the satisfiability problem for programs without disjunctions in ASP (Dantsin *et al*. 2001). All completeness results in this section hold under *logarithmic space reductions*. Recall that a *logarithmic space reduction* from a language A to a language B is a function f that can be computed by a Turing machine using logarithmic space such that every string x satisfies $x \in A$ if and only if $f(x) \in B$. A problem A is *hard* for a complexity class C if every problem in C can be reduced to A by a logarithmic space reduction. A problem A is *complete* for a complexity class C if it is hard for C and it is in C.

[5] See, for example Papadimitriou (1994) for a comprehensive treatment of this subject.

https://doi.org/10.1017/S1471068426100647 Published online by Cambridge University Press

Corollary 2.
Let $\mathfrak{T}$ be a theory that is decidable in polynomial time. Then, the satisfiability problem for $\mathcal{T}$-program s with theory $\mathfrak{T}$ is NP*-complete.*

Finally, we obtain the following results for the complexity of checking whether two $\mathcal{T}$-program s are strongly equivalent.

Theorem 8.
Deciding whether two $\mathcal{T}$-programs are strongly equivalent is undecidable, even if both programs are restricted to be single facts.

Theorem 9.
Deciding whether two $\mathcal{T}$-programs with a theory $\mathfrak{T}$ are strongly equivalent is decidable in $\mathsf{coNP}^{\mathsf{O}}$, where O is an oracle that solves the satisfiability problem for the external theory $\mathfrak{T}$.

Since regular ASP programs are a particular case of $\mathcal{T}$-programs, the following result immediately follows from the co-NP-completeness of the strongly equivalence problem for ASP problems (Lin 2002, Corollary 2.1).[6]

Corollary 3.
Let $\mathfrak{T}$ be a theory that is decidable in polynomial time. Then, deciding whether two $\mathcal{T}$-program s are strongly equivalent is coNP*-complete.*

10 Conclusion

We have introduced a method to establish whether two logic programs with constraints are strongly equivalent. This method is based on a translation of logic programs with constraints into HT_c-theories and the characterization of strong equivalence in the context of HT_c. In particular, we have shown that two logic programs with constraints are strongly equivalent if and only if their translations are equivalent in HT_c and study the computational complexity of this problem.

The translation from logic programs with constraints into HT_c-theories is of independent interest as it can serve as a tool to study strong equivalence of other extensions of logic program with constraints. It is worth recalling that HT_c was introduced as a generalization of HT to deal with CSPs involving defaults over the variables of the external theory (Cabalar *et al*. 2016). Since then, HT_c has been extended to also formalize the semantics of aggregates over those same variables (Cabalar *et al*. 2020a, 2020b). As a future work, we are planning to leverage this translation to implement a new hybrid solver that can handle these two features (constraints with defaults and aggregates) and uses `clingo 5` as a back-end solver.

Supplementary material

To view supplementary material for this article, please visit https://doi.org/10.1017/S1471068426100647.

[6] Lin (2002) does not explicitly state the kind of reduction used, but a careful read of the proof reveals that it is a logarithmic space reduction.

https://doi.org/10.1017/S1471068426100647 Published online by Cambridge University Press

References

Banbara, M., Kaufmann, B., Ostrowski, M. and Schaub, T. 2017. Clingcon: The next generation. *Theory and Practice of Logic Programming* 17, 4, 408–461. https://doi.org/10.1017/S1471068417000138.

Cabalar, P., Fandinno, J., Schaub, T. and Wanko, P. 2020a. An ASP semantics for constraints involving conditional aggregates. In *Proceedings of the Twenty-Fourth European Conference on Artificial Intelligence (ECAI'20)*, G. De Giacomo, A. Catalá, B. Dilkina, M. Milano, S. Barro, A. Bugarín and J. Lang, Eds. IOS Press, 664–671. https://doi.org/10.3233/FAIA200152.

Cabalar, P., Fandinno, J., Schaub, T. and Wanko, P. 2020b. A uniform treatment of aggregates and constraints in hybrid ASP. In *Proceedings of the Seventeenth International Conference on Principles of Knowledge Representation and Reasoning (KR'20)*, D. Calvanese, E. Erdem and M. Thielscher, Eds. AAAI Press, 193–202. https://doi.org/10.24963/KR.2020/20.

Cabalar, P., Fandinno, J., Schaub, T. and Wanko, P.2023. On the semantics of hybrid ASP systems based on clingo. *Algorithms* 16, 4. https://doi.org/10.3390/a16040185.

Cabalar, P., Kaminski, R., Ostrowski, M. and Schaub, T. 2016. An ASP semantics for default reasoning with constraints. In *Proceedings of the Twenty-Fifth International Joint Conference On Artificial Intelligence (IJCAI'16)*, S. Kambhampati, Ed. IJCAI/AAAI Press, 1015–1021. https://doi.org/10.5555/3060621.3060762.

Dantsin, E., Eiter, T., Gottlob, G. and Voronkov, A. 2001. Complexity and expressive power of logic programming. *ACM Computing Surveys* 33, 3, 374–425.

Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T. and Wanko, P. 2016. Theory solving made easy with clingo 5. In *Technical Communications of the Thirty-Second International Conference on Logic Programming (ICLP'16)*, M. Carro, A. King, N. Saeedloei and M. De Vos, Eds. Open Access Series in Informatics (OASIcs), vol. 52. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2:1–2:15.

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference and Symposium of Logic Programming (ICLP'88)*, R. Kowalski and K. Bowen, Eds. MIT Press, 1070–1080. https://doi.org/10.1201/b10397-6.

Heyting, A. 1930. Die formalen Regeln der intuitionistischen Logik. In *Sitzungsberichte der Preussischen Akademie der Wissenschaften*. Deutsche Akademie der Wissenschaften zu Berlin, 42–56.

Janhunen, T., Kaminski, R., Ostrowski, M., Schellhorn, S., Wanko, P. and Schaub, T. 2017. Clingo goes linear constraints over reals and integers. *Theory and Practice of Logic Programming* 17, 5–6, 872–888. https://doi.org/10.1017/S1471068417000242.

Lierler, Y. 2023. Constraint answer set programming: Integrational and translational (or SMT-based) approaches. *Theory and Practice of Logic Programming* 23, 1, 195–225. https://doi.org/10.1017/S1471068421000478.

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

Lifschitz, V., Pearce, D. and Valverde, A. 2001. Strongly equivalent logic programs. *ACM Transactions on Computational Logic* 2, 4, 526–541. https://doi.org/10.1145/383779.383783.

Lin, F. 2002. Reducing strong equivalence of logic programs to entailment in classical propositional logic. In *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR'02)*, D. Fensel, F. Giunchiglia, D. McGuinness and M. Williams, Eds. Morgan Kaufmann Publishers, 170–176.

https://doi.org/10.1017/S1471068426100647 Published online by Cambridge University Press

Matiyasevich, Y. 1972. Diophantine sets. *Russian Mathematical Surveys* 27, 5, 124–164.

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.

Papadimitriou, C. 1994. *Computational Complexity*. Addison-Wesley.

Pearce, D. 1997. A new logical characterisation of stable models and answer sets. In *Proceedings of the Sixth International Workshop on Non-Monotonic Extensions of Logic Programming (NMELP'96)*, J. Dix, L. Pereira and T. Przymusinski, Eds. Lecture Notes in Computer Science, vol. 1216. Springer-Verlag, 57–70. https://doi.org/10.1007/BFb0023801.

Pearce, D. 2006. Equilibrium logic. *Annals of Mathematics and Artificial Intelligence* 47, 1–2, 3–41. https://doi.org/10.1007/s10472-006-9028-z.

https://doi.org/10.1017/S1471068426100647 Published online by Cambridge University Press