A Semantic Perspective on Omission Abstraction in ASP[*]

Zeynep G. Saribatur , **Thomas Eiter**
Institute of Logic and Computation, TU Wien, Austria
{zeynep, eiter}@kr.tuwien.ac.at

Abstract

The recently introduced notion of ASP abstraction is on reducing the vocabulary of a program while ensuring over-approximation of its answer sets, with a focus on having a syntactic operator that constructs an abstract program. It has been shown that such a notion has the potential for program analysis at the abstract level by getting rid of irrelevant details to problem solving while preserving the structure, that aids in the explanation of the solutions. We take here a further look on ASP abstraction, focusing on abstraction by omission with the aim to obtain a better understanding of the notion. We distinguish the key conditions for omission abstraction which sheds light on the differences to the well-studied notion of forgetting. We demonstrate how omission abstraction fits into the overall spectrum, by also investigating its behavior in the semantics of a program in the framework of HT logic.

1 Introduction

Abstraction for ASP (Saribatur and Eiter 2018; Saribatur et al. 2019) is a method whose main aim is to achieve an *over-approximation* of a program by reducing the vocabulary while keeping the rules; this is valuable to aid tasks like explanation finding or showing solutions that omit detail. It was approached from the syntactic level, with the aim to adjust the rules in a way that the original answer sets still have a representative in the abstract program while the structure of the original program is preserved as much as possible. Inspired by the seminal work in model checking (Clarke et al. 2003) the focus so far has been on starting with a coarse abstraction of the program and refining it until a concrete solution is encountered, where at any time we have an abstract program at hand that yields solutions without unnecessary details that are omitted.

The well-studied concept of forgetting (Gonçalves et al. 2017) on the other hand is about preserving the semantics of the original program exactly, by adhering to a number of properties that have been investigated over the years. Recently, a syntactic forgetting operator has been introduced (Berthold et al. 2019) which achieves the key property of *strong persistence* (**SP**) whenever possible, at the expense of preserving the original syntax. In terms of forgetting, the notion of over-approximation matches the *weak consequence* (**wC**) property, which did not receive much attention as it is in contrast with the main goal of forgetting. Our previous investigations showed that a more relaxed abstraction approach and a methodology for refining an abstraction if needed can achieve valuable results, especially in singling out the relevant parts of a program for decision-making (Saribatur and Eiter 2018; Saribatur et al. 2019; Eiter et al. 2019).

In this short paper, we review omission abstraction in the light of forgetting. To clarify some differences and to get some better understanding of abstraction by omission (Saribatur and Eiter 2018), we present desired properties that any operator should satisfy; as we show, the previously introduced operator *omit* is (under some trivial proviso) optimal when adhering to the properties and we show how the notion fits into the overall picture. We notably observe that some of the forgetting operators can also count as omission abstraction, when structure preservation and modularity is not important. We also explore the behavior of omission abstraction through the semantics of a program by focusing on HT-logic, which is regarded as the monotonic core of ASP.

This work sets a starting point for further research on exploring the use of ASP abstraction by taking a semantic view on the notion; notably, the semantic view has been fruitfully driving research on forgetting in the last decade. We provide some results as a base for extensions and follow up work.

2 Background

ASP A logic program P over a set $\mathcal{A}$ of propositional atoms is a set of rules r of the form

$$\alpha_0 \leftarrow \alpha_1, \dots, \alpha_m, not\ \alpha_{m+1}, \dots, not\ \alpha_n,\ \ 0 \leq m \leq n,$$

where each $\alpha_i \in \mathcal{A}$ is a propositional literal and not is default negation; r is a *constraint* if α_0 is falsity ($\perp$, then omitted) and a *fact* if $n = 0$. We also write $H(r) \leftarrow B(r)$ or $H(r) \leftarrow B^+(r),\ not\ B^-(r)$, where $H(r) = \alpha_o$ and $B^+(r) = \{\alpha_1, \dots, \alpha_m\}$ is the positive body and $B^-(r) = \{\alpha_{m+1}, \dots, \alpha_n\}$ the negative body. Furthermore, we let $B^\pm(r) = B^+(r) \cup B^-(r)$. We occasionally omit r from $B^\pm(r), B(r)$ etc. if r is understood. The *GL-reduct* is given by $P^I = \{\alpha_0 \leftarrow B^+(r) \mid r \in P, B^-(r) \cap I = \emptyset\}$. An interpretation I is an *answer set*, if it is a minimal model of P^I. We denote the set of all answer sets by $AS(P)$. For a

[*]Extended version can be found at http://www.kr.tuwien.ac.at/research/reports/rr2001.pdf

set $S \subseteq \mathcal{A}$ of atoms, $S_{|A}$ denotes the projection to the atoms in A and $\overline{S}$ is a shorthand for $\mathcal{A} \setminus S$. As a common syntactic extension, we consider choice rules of the form $\{\alpha\} \leftarrow B$, which according to the ASP Core-2 recommendation (Calimeri et al. 2020) stands for the rules $\alpha \leftarrow \mathit{not}\, \alpha', B$ and $\alpha' \leftarrow \mathit{not}\, \alpha, B$ where α' is a fresh (hidden) atom. An answer set I is then identified with the interpretation obtained from I by projecting off auxiliary atoms α'. We alternatively may write $\{\alpha\} \leftarrow B$ as $\alpha \leftarrow \mathit{not}\, \mathit{not}\, \alpha, B$, i.e. use double negation; the results of HT-Logic extend to the use of double negation. An *operator* over a class $\mathcal{C}$ of programs over $\mathcal{A}$ is a partial function $f : \mathcal{C} \times 2^{\mathcal{A}} \rightarrow \mathcal{C}$, where $f(P, A)$ is the *result* of applying f on a program P by considering $A \subseteq \mathcal{A}$.

HT-models An *HT-interpretation* is a pair $\langle X, Y \rangle$ such that $X \subseteq Y \subseteq \mathcal{A}$; it is *total* if $X = Y$ and *non-total* otherwise. An HT-interpretation $\langle X, Y \rangle$ is an *HT-model* of a program P if $Y \models P$ and $X \models P^Y$. The set of all HT-models of P is denoted by $\mathcal{HT}(P)$. A set Y of atoms is an answer set of P if $\langle Y, Y \rangle \in \mathcal{HT}(P)$ and no non-total $\langle X, Y \rangle \in \mathcal{HT}(P)$ exists. Two programs P_1, P_2 are *equivalent* if $AS(P_1) = AS(P_2)$, and *strongly equivalent*, denoted by $P_1 \equiv P_2$, if $AS(P_1 \cup R) = AS(P_2 \cup R)$ for every R over $\mathcal{A}$. As well-known, $P_1 \equiv P_2$ amounts to $\mathcal{HT}(P_1) = \mathcal{HT}(P_2)$ (Lifschitz, Pearce, and Valverde 2001).

2.1 Omission Abstraction

We call a program P' over $\mathcal{A}'$ an *abstraction* of a program P over $\mathcal{A}$, if there exists a mapping $m : \mathcal{A} \rightarrow \mathcal{A}' \cup \{\top\}$ such that for each answer set I of P, $I' = \{m(l) \mid l \in I\}$ is an answer set of P'. In (Saribatur and Eiter 2018) we introduced the notion of *omission-based abstraction* which relies on a *mapping* $m_A : \mathcal{A} \rightarrow \mathcal{A} \cup \{\top\}$ for a set A of atoms to be omitted, such that $m_A(\alpha) = \top$ if $\alpha \in A$ and $m_A(\alpha) = \alpha$ if $\alpha \in \mathcal{A} \setminus A$.

Given an original program P and a set A of atoms to be omitted, an abstract program $\mathit{omit}(P, A)$ is constructed as follows. For every rule $r : \alpha \leftarrow B$ in P, we have

$$\mathit{omit}(r, A) = \begin{cases} r, & \text{if } A \cap B^{\pm} = \emptyset, \alpha \notin A, \\ \{\alpha\} \leftarrow m_A(B), & \text{if } A \cap B^{\pm} \neq \emptyset, \alpha \notin A \cup \{\bot\}, \\ \emptyset, & \text{otherwise.} \end{cases}$$

where $m_A(B)$ stands for $B^+(r) \setminus A, \mathit{not}\ (B^-(r) \setminus A)$ which projects away the omitted atoms. We showed that $\mathit{omit}(P, A)$ is an over-approximation of P. However, spurious answer sets may appear in $\mathit{omit}(P, A)$, i.e., answer sets which can not be mapped back to some original answer set.

Example 1. *Consider the program P_1 and the resulting abstract programs after applying $\mathit{omit}(P_1, A)$ for $A = \{a, c\}$ and $A = \{b\}$ shown below.*

P_1	$\mathit{omit}(P_1, \{a, c\})$	$\mathit{omit}(P_1, \{b\})$
$c \leftarrow \mathit{not}\ d.$		$c \leftarrow \mathit{not}\ d.$
$d \leftarrow \mathit{not}\ c.$	$\{d\}.$	$d \leftarrow \mathit{not}\ c.$
$a \leftarrow \mathit{not}\ b, c.$		$\{a\} \leftarrow c.$
$b \leftarrow d.$	$b \leftarrow d.$	
$\{\{c, a\}, \{d, b\}\}$	$\{\{\}, \{d, b\}\}$	$\{\{c\}, \{c, a\}, \{d\}\}$

Every answer set of P_1 can be mapped to some answer set of $\mathit{omit}(P_1, \{a, c\})$ resp. $\mathit{omit}(P_1, \{b\})$ if the omitted literals are projected away, i.e., $AS(P_1)_{|\overline{A}} \subseteq AS(\mathit{omit}(P_1, A))$. Note that $\mathit{omit}(P_1, \{b\})$ also has a spurious answer set $\{c\}$, as it can not be mapped back to some original answer set.

We call an abstraction *faithful* if the constructed abstract program has no spurious answer sets. A faithful abstraction is referred as *refinement-safe* if all abstractions achievable by adding back some omitted atoms are also faithful. We remark that not every faithful abstraction can be refinement-safe, e.g., $\mathit{omit}(P, \mathcal{A}) = \emptyset$ is faithful (whenever P is satisfiable), but adding back some atoms may reach an abstraction with a spurious answer set.

3 Desiderata for Omission Abstraction

As mentioned above, abstraction aims at omitting details from a program to simplify matters, while structure should be preserved. Many operators $f_o(P, V)$ can be imagined that omit atoms V from a program P, yielding another program in which no atom from V occurs. In order to assess whether such f_o can be regarded as an omission abstraction, we consider the following desired properties:

(D1) Over-approximation should be achieved, i.e., $AS(f_o(P, V)) \supseteq AS(P)|_{\overline{V}}$ holds.

(D2) Rules not involving atoms to omit must be preserved, i.e., $\{r \in P \mid r \text{ is over } \overline{V}\} \subseteq f_o(P, V)$.

(D3) New rules should be introduced only if P contains atoms to omit, i.e., if P is over $\overline{V}$, then $f_o(P, V) \subseteq P$.

Here **(D1)** is the semantic key requirement that no answer set of the original program is lost in the abstraction; **(D2)**-**(D3)** ensure that only modifications relevant to the abstraction are made. To preserve the structure of rules containing atoms to be omitted and to ensure that rules are not changed to arbitrary ones, the next condition should be satisfied.

(D4) Rules in $f_o(P, V)$ should be obtained by strengthening / weakening rules r in P, meaning that

(i) literals can only be added in the body of r, or
(ii) literals can only be removed from the body of r, or
(iii) the head of r can be modified.[1]

The next conditions are important for the incrementality of the operator, which would be beneficial for the computation of the omission.

(D5) Modularity: $f_o(P, V) = \bigcup_{r \in P} f_o(r, V)$.

(D6) Iteration: for $V = \{v_1, ..., v_n\}$,

$$f_o(f_o(f_o(P, v_1), v_2)..., v_n) = f_o(P, V).$$

Conditions **(D5)**-**(D6)** allow for applying the omission operator incrementally by using the previously computed abstraction to compute the next one, rule by rule resp. atom by atom. In particular, **(D5)** ensures that we can expand the program without touching the previously abstracted parts.

[1]Item (iii) may be more restrictive, and e.g. allow only for the change into a choice.

Among the different operators that can satisfy **(D1)-(D5)**, with a slight modification, *omit* becomes the operator that achieves the tightest possible abstract program, with as few spurious answer sets as possible, even if condition **(D4)** is disregarded. To show this, we denote by F_o the class of omission operators that satisfy **(D1)-(D3)** and **(D5)**, and define an ordering $\preceq$ on F_o such that $f_o \preceq f_{o'}$ for $f_o, f_{o'} \in F_o$ if $AS(f_{o'}(P,V)) \supseteq AS(f_o(P,V))$ for every program P and V, i.e., f_o introduces less spurious answer sets than $f_{o'}$.

In order to avoid introducing guesses for tautologic rules such as $a \leftarrow d, b, \mathit{not}\ b$ where b is to be omitted, let the omission operator omit^+ be defined as omit except that such tautologic rules are skipped. We then have the following.

Theorem 1. *The operator omit^+ is a minimal operator w.r.t. $\preceq$ in F_o. Moreover, omit^+ is the unique such operator under strong equivalence, i.e., every $\preceq$-minimal $f_o \in F_o$ satisfies $\mathit{omit}^+(P,V) \equiv f_o(P,V)$ for all values of P and V.*

This result shows that omit^+ is optimal, i.e., there is no tighter way of doing abstraction while adhering to **(D1)-(D3)** and **(D5)**, and provides us via **(D4)** also with a canonical form. To achieve faithfulness in general, necessarily some non-modular operations will be needed.

Furthermore, *omit* is closed in the class of normal programs (when choice rules are written through two auxiliary rules), thus making it possible to be iterated **(D6)**.

4 Through the Lens of Forgetting

We refer to (Gonçalves, Knorr, and Leite 2016a; Delgrande 2017) for recent surveys on forgetting and just shortly summarize the notions needed here. Below are some of the properties considered in forgetting, where F is a class of forgetting operators and $\mathcal{C}$ a class of programs:

(wC) F satisfies *weakened Consequence* if, for each $f \in F$, $P \in \mathcal{C}$ and $V \subseteq \mathcal{A}$, we have $AS(P)_{|\overline{V}} \subseteq AS(f(P,V))$.

(SI) F satisfies *Strong (addition) Invariance* if, for each $f \in F$, $P \in \mathcal{C}$ and $V \subseteq \mathcal{A}$, we have $f(P,V) \cup R \equiv f(P \cup R, V)$ for all programs $R \in \mathcal{C}$ over $\overline{V}$.

(CP) F satisfies *Consequence Persistence* if, for each $f \in F$, $P \in \mathcal{C}$ and $V \subseteq \mathcal{A}$, we have $AS(f(P,V)) = AS(P)_{|\overline{V}}$.

(SP) F satisfies *Strong Persistence* if, for each $f \in F$, $P \in \mathcal{C}$ and $V \subseteq \mathcal{A}$, we have $AS(f(P,V) \cup R) = AS(P \cup R)_{|\overline{V}}$ for all programs $R \in \mathcal{C}$ over $\overline{V}$.

Strong persistence is also considered by Gonçalves, Knorr, and Leite (2016b) for a particular forgetting instance $\langle P, V\rangle$ where P is a program in $\mathcal{C}$ and $V \subseteq \mathcal{A}$, denoted by **(SP)**$_{\langle P,V\rangle}$, which holds if $AS(f(P,V) \cup R) = AS(P \cup R)_{|\overline{V}}$, for all programs $R \in \mathcal{C}$ over $\overline{V}$.They also introduced a criterion Ω to characterize the instances for which an operator achieving **(SP)**$_{\langle P,V\rangle}$ is impossible.

Definition 1. *(Gonçalves, Knorr, and Leite 2016b) Let P be a program over $\mathcal{A}$ and $V \subseteq \mathcal{A}$. An instance $\langle P, V\rangle$ satisfies criterion Ω if there exists $Y \subseteq \mathcal{A} \setminus V$ such that the set of sets*

$$\mathcal{R}^{Y}_{\langle P,V\rangle} = \{R^{Y,A}_{\langle P,V\rangle} \mid A \in \mathit{Rel}^{Y}_{\langle P,V\rangle}\}$$

is non-empty and has no least element, where

$$R^{Y,A}_{\langle P,V\rangle} = \{X \setminus V \mid \langle X, Y \cup A\rangle \in \mathcal{HT}(P)\}$$

$$\begin{aligned}\mathit{Rel}^{Y}_{\langle P,V\rangle} = \{A \subseteq V \mid {} & \langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}(P) \textit{ and} \\ & \nexists A' \subset A \textit{ s.t. } \langle Y \cup A', Y \cup A\rangle \in \mathcal{HT}(P)\}.\end{aligned}$$

It was shown that it is not possible to forget about V from P while satisfying strong persistence exactly when $\langle P, V\rangle$ satisfies criterion Ω. Later, when investigating further uses of forgetting when Ω is satisfied, Gonçalves et al. (2017) considered two relaxed properties:

(sSP) F satisfies *strengthened Strong Persistence* if, for each $f \in F$, $P \in \mathcal{C}$ and $V \subseteq \mathcal{A}$, we have $AS(f(P,V) \cup R) \subseteq AS(P \cup R)_{|\overline{V}}$ for all programs $R \in \mathcal{C}$ over $\overline{V}$.

(wSP) F satisfies *weakened Strong Persistence* if, for each $f \in F$, $P \in \mathcal{C}$ and $V \subseteq \mathcal{A}$, we have $AS(P \cup R)_{|\overline{V}} \subseteq AS(f(P,V) \cup R)$ for all programs $R \in \mathcal{C}$ over $\overline{V}$.

They correspond to under- and over-approximation of the answer sets, respectively. As easily seen,

Proposition 2. **(wSP)** *is a consequence of* **(wC)** *and* **(SI)**.

Among the forgetting operators, F_{SP} (Gonçalves, Knorr, and Leite 2016b) and F_{Sas} (Knorr and Alferes 2014) satisfy both **(wC)** and **(SI)**. These operators can also be considered as omission abstraction since they are in line with **(D1)-(D3)**. However, the syntactic conditions **(D4)-(D6)** do not hold in general, and it remains unclear whether they will be achieved by particular operators. Iteration of the operator within the class of normal programs might not be possible as these operators are not closed.

4.1 Putting Omission Abstraction in this Picture

Here, we initiate the search for the relation between some of the properties that have been considered for omission abstraction and for forgetting, respectively. We can see that **(D1)-(D3)** with **(D5)** immediately allow us to conclude the following.

Proposition 3. F_o *satisfies* **(wC)** and **(SI)**.

Since omission abstraction preserves all the rules that do not mention the set V of atoms to be omitted, in fact a stronger version of **(SI)** is satisfied.

Proposition 4. *For every $f_o \in F_o$, we have $f_o(P \cup R, V) = f_o(P,V) \cup R$ for all programs R over $\overline{V}$.*

We can also see that the weakened version of strong persistence is also always satisfied.

Proposition 5 ((wSP)). *For every $f_o \in F_o$, $AS(f_o(P,V) \cup R) \supseteq AS(P \cup R)_{|\overline{V}}$ holds for all programs $R \in \mathcal{C}$ over $\overline{V}$.*

Faithfulness is a valuable property for abstraction, as it keeps the relevant part of the program needed for reaching a concrete result. However, it is not always possible to achieve a faithful abstraction under modularity, thus the property **(CP)** can not be satisfied in general; for a particular instance $\langle P, V\rangle$, denoted **(CP)**$_{\langle P,V\rangle}$, it is achievable.

Proposition 6. *For every $f_o \in F_o$, faithful omission abstraction $f_o(P,V)$ satisfies* **(CP)**$_{\langle P,V\rangle}$.

It was shown that **(CP)** and **(SI)** together are equivalent to **(SP)** (Gonçalves, Knorr, and Leite 2016a). However, this is not true for particular instances $\langle P, V\rangle$, thus we can not talk about directly achieving **(SP)**$_{\langle P,V\rangle}$ from **(CP)**$_{\langle P,V\rangle}$. However, by Proposition 6, knowing that **(CP)**$_{\langle P,V\rangle}$ holds, one can infer that spuriousness occurs as the added program R interferes with the rules that were changed in the abstraction $f_o(P, V)$. Diagnosing the necessary interaction of changed and added rules when **(SP)**$_{\langle P,V\rangle}$ fails while **(CP)**$_{\langle P,V\rangle}$ holds can serve as a basis for modifying an omission operator, in particular on a restricted program class.

Nevertheless, it is also possible to achieve **(SP)**$_{\langle P,V\rangle}$ for some $\langle P, V\rangle$ with F_o. For this, we use the class F_M of forgetting operators in (Gonçalves, Knorr, and Leite 2016b), which focuses on constructing programs with HT-models $\langle X, Y\rangle$ where X is in the intersection $\bigcap \mathcal{R}^Y_{\langle P,V\rangle}$ or in the union $\bigcup \mathcal{R}^Y_{\langle P,V\rangle}$. It was shown that F_M satisfies **(wC)**, **(CP)**, **(sSP)**, but not **(SI)**. Thus, by Proposition 5 we obtain

Proposition 7. *For $f_o \in F_o$, if $\mathcal{HT}(f_o(P, V)) = \mathcal{HT}(f(P, V))$ for $f \in F_M$, then $AS(f(P, V) \cup R) = AS(P \cup R)_{|\overline{V}}$ for all programs $R \in \mathcal{C}$ over $\overline{V}$.*

This shows that if omission abstraction f_o (which satisfies **(wSP)**) obtains the same HT models as the operators in F_M (which satisfy **(sSP)**), we can say that **(SP)**$_{\langle P,V\rangle}$ is achieved. This understanding can lead to finding a new set of operators achieving the desired properties from both sides.

Example 2. *The HT-models of the abstract program $omit(P_1, \{a, c\})$ from Example 1 are $\langle\emptyset, \emptyset\rangle, \langle\emptyset, b\rangle, \langle b, b\rangle, \langle db, db\rangle$ which is equivalent to $\mathcal{HT}(f(P_1, \{a, c\}))$ for $f \in F_M$, and thus $omit(P_1, \{a, c\})$ satisfies* **(SP)**$_{\langle P_1,\{a,c\}\rangle}$.

Furthermore, given that Gonçalves, Knorr, and Leite (2016b) proved that **(SP)**$_{\langle P,V\rangle}$ is not possible if $\langle P, V\rangle$ satisfies Ω, we conclude the following.

Proposition 8. *Let $f_o \in F_o$. If some $f \in F_M$ exists such that $\mathcal{HT}(f_o(P, V)) = \mathcal{HT}(f(P, V))$, then $\langle P, V\rangle$ does not satisfy Ω.*

For some operators, when incrementally omitting a set of atoms while achieving abstractions that satisfy **(wC)**, it may be possible to eventually achieve an abstraction that satisfies strong persistence. For operator $omit$, however, we can conclude from our results in the next section that if after omitting some atoms, **(wC)** holds but not **(CP)**$_{\langle P,V\rangle}$, then **(SP)**$_{\langle P,V\rangle}$ can not be achieved after omitting further atoms.

Note that even when **(SP)**$_{\langle P,V\rangle}$ is not possible, it would still be possible to achieve **(CP)**$_{\langle P,V\rangle}$.

5 Focusing on HT Logic

In this section, we take a deeper look into the semantics of abstraction through the HT logic,[2] by investigating the behavior of the omission operator $omit$, to get a better understanding of the concept. Note that the omission operator $omit$ can be extended to handle rules with double negation; details can be found in the extended version.

[2]We write choice rules $\{a\} \leftarrow B$ as $a \leftarrow \mathit{not}\, \mathit{not}\, a, B$. confining double negation to this use, and constraints $\bot \leftarrow a, B$. as $f \leftarrow a, \mathit{not}\ f, B$, for a handle on all possible original HT-models.

The first observation is on preserving the total models in the abstraction.

Proposition 9. *If $\langle Y, Y\rangle \in \mathcal{HT}(P)$ then $\langle Y, Y\rangle_{|\overline{V}} \in \mathcal{HT}(omit(P, V))$.*

Proof. Assume $\langle Y \setminus V, Y \setminus V\rangle \notin \mathcal{HT}(omit(P, V))$. This means $Y \setminus V \models B(\hat{r})$ but $Y \setminus V \not\models H(\hat{r})$ for some $\hat{r} \in omit(P, V)$. So $\hat{r}$ cannot be a choice rule. However $\hat{r}$ also cannot be an unchanged rule as it would contradict $\langle Y, Y\rangle \in \mathcal{HT}(P)$. □

Although all total models $\langle Y, Y\rangle$ are preserved in $\mathcal{HT}(omit(P, V))$, not all non-total $\langle X, Y\rangle \in \mathcal{HT}(P)$ might be preserved, thus causing spurious answer sets in $omit(P, V)$.

The next proposition tells us that if setting all omit atoms V to true resp. false preserves an HT-model, then setting all to false yields an abstract HT-model.

Proposition 10. *Let $\langle X, Y\rangle \in \mathcal{HT}(P)$. If $\langle X\cup V, Y\cup V\rangle \in \mathcal{HT}(P)$ and $\langle X\setminus V, Y\setminus V\rangle \in \mathcal{HT}(P)$, then $\langle X\setminus V, Y\setminus V\rangle \in \mathcal{HT}(omit(P, V))$.*

Unfortunately, $\mathcal{HT}(omit(P, V)) \subseteq \mathcal{HT}(P)_{|\overline{V}}$ does not always hold; sometimes, "new" HT-models could be introduced with the abstraction.

Definition 2 (spurious HT-model)**.** *An HT-model $\langle X, Y\rangle \in \mathcal{HT}(omit(P, V))$ is* spurious*, if no $\langle X', Y'\rangle \in \mathcal{HT}(P)$ exists s.t. $X' \setminus V = X$, and $Y' \setminus V = Y$.*

As regards spurious total models, they can occur only if rules change in the abstract program as follows.

Proposition 11. *If $\langle Y, Y\rangle \in \mathcal{HT}(omit(P, V))$ is spurious, then some rule $r \in P$ satisfies $B^+(r) \cap V \neq \emptyset$ and $H(r) \notin V$.*

Note that a spurious total HT-model in the abstract program does not necessarily mean that there are spurious answer sets, since also further spurious non-total models can occur. However, the following result shows that spurious non-total models occur only for spurious total models.

Proposition 12. *If $\langle X, Y\rangle \in \mathcal{HT}(omit(P, V))$ is spurious, then $\langle Y, Y\rangle \in \mathcal{HT}(omit(P, V))$ is also spurious.*

Based on this, we then obtain the following property.

Proposition 13. *$omit(P, V)$ has no spurious total HT-models iff $\mathcal{HT}(omit(P, V)) \subseteq \mathcal{HT}(P)_{|\overline{V}}$.*

We can also characterize the definition of faithfulness of an abstraction $omit(P, V)$ (i.e., **(CP)**$_{\langle P,V\rangle}$) through the HT semantics.

Proposition 14 (Faithfulness via HT semantics)**.** *For a program P and atoms V, $omit(P, V)$ is faithful iff we have:*

1. *for all $\langle Y, Y\rangle$, $\langle X, Y\rangle \in \mathcal{HT}(P)$ where $X \subset Y$, some non-total $\langle X', Y \setminus V\rangle \in \mathcal{HT}(omit(P, V))$ exists; and*
2. *for all $\langle Y, Y\rangle \in \mathcal{HT}(omit(P, V))$ s.t. for every $A' \subseteq V$ where $\langle Y \cup A', Y \cup A'\rangle \notin \mathcal{HT}(P)$ some non-total $\langle X, Y\rangle \in \mathcal{HT}(omit(P, V))$ exists.*

Condition 1 ensures that all HT-models that are not answer sets do not become answer sets in the abstraction, while condition 2 ensures that for new total HT-models some "killer models" exist eliminating them as answer sets.

From Prop. 9-12 and 13 we get the following result.

Theorem 15. *If $omit(P, V)$ is faithful and satisfies $\mathcal{HT}(omit(P, V)) \subseteq \mathcal{HT}(P)_{|\overline{V}}$, then $omit(P, V')$ is faithful for every $V' \subseteq V$.*

Notice that this result also extends to the case whenever $omit(P, V)$ satisfies **(SP)**$_{\langle P,V\rangle}$.

This result shows that if $omit(P, V)$ satisfies the conditions in Theorem 15, then it would be possible to add back some of the omitted atoms to the program, while preserving faithfulness. Notice that this coincides with our notion of refinement-safe faithfulness. This notion helps to avoid the cases of having faithful abstractions that omit too many details but no longer remain faithful when some of the details are added back. Another way of viewing refinement-safety is from the other way around. Atoms can be omitted one-by-one, while still preserving the decisions until omission can no longer be made without introducing spuriousness. Our result gives a semantic condition which is sufficient but not necessary for obtaining refinement-safety.

6 Conclusion

We have reviewed omission abstraction in the light of recent work on forgetting from logic programs. To understand the difference, we have described desired properties for the former, and we showed optimality of the *omit* operator; some forgetting operators can be viewed as achieving omission abstraction while not adhering to modularity and incrementality in general. It remains to extend this work and chart classes of programs where forgetting operators satisfy the latter. In that, the modularity condition, which was the starting criteria for abstraction, may be weakened so that a program can be partitioned into subprograms for independent omission while still maintaining the results in Section 4.1. The longer range aim is to recognize how omission abstraction aligns with the forgetting spectrum, and how results and tools of the latter may be used to construct omission abstraction operators with certain properties. Moreover, the concept of omission abstraction and its possible applications may bring a fresh perspective to research in forgetting.

Our semantic look at the omission abstraction operator *omit* in the well-established framework of HT-logic provides a base for future investigations towards understanding ASP abstraction and its desired properties. A characterization of faithfulness (in particular of refinement-safe faithfulness) would be interesting, yet we expect it to be more involved. This should be even more true for a relativized version of faithfulness, in which auxiliary atoms (as customary in ASP encodings) are disregarded, making it more suitable for applications; its linkage to the forgetting framework also remains to be studied.

Acknowledgments

We thank João Leite for his questions on the possible relations of forgetting to abstraction and for pointing to the references. Furthermore, we are grateful to the reviewers for their helpful and constructive comments.

References

Berthold, M.; Gonçalves, R.; Knorr, M.; and Leite, J. 2019. A syntactic operator for forgetting that satisfies strong persistence. *Theory and Practice of Logic Programming* 19(5-6):1038–1055.

Calimeri, F.; Faber, W.; Gebser, M.; Ianni, G.; Kaminski, R.; Krennwallner, T.; Leone, N.; Maratea, M.; Ricca, F.; and Schaub, T. 2020. Asp-core-2 input language format. *Theory and Practice of Logic Programming* 20(2):294–309.

Clarke, E.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2003. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM* 50(5):752–794.

Delgrande, J. P. 2017. A knowledge level account of forgetting. *Journal of Artificial Intelligence Research* 60:1165–1213.

Eiter, T.; Saribatur, Z. G.; and Schüller, P. 2019. Abstraction for zooming-in to unsolvability reasons of grid-cell problems. In *Proc. of the IJCAI 2019 Workshop on Explainable Artificial Intelligence (XAI)*.

Gonçalves, R.; Knorr, M.; Leite, J.; and Woltran, S. 2017. When you must forget: Beyond strong persistence when forgetting in answer set programming. *Theory and Practice of Logic Programming* 17(5-6):837–854.

Gonçalves, R.; Knorr, M.; and Leite, J. 2016a. The ultimate guide to forgetting in answer set programming. In *Proc. of the 15th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 135–144.

Gonçalves, R.; Knorr, M.; and Leite, J. 2016b. You can't always forget what you want: On the limits of forgetting in answer set programming. In *Proc. of the 22nd European Conference on Artificial Intelligence (ECAI)*, 957–965. IOS Press.

Knorr, M., and Alferes, J. J. 2014. Preserving strong equivalence while forgetting. In *Proc. of the 14th European Conference on Logics in Artificial Intelligence (JELIA)*, 412–425.

Lifschitz, V.; Pearce, D.; and Valverde, A. 2001. Strongly equivalent logic programs. *ACM Transactions on Computational Logic* 2(4):526–541.

Saribatur, Z. G., and Eiter, T. 2018. Omission-based abstraction for answer set programs. In *Proc. of the 16th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 42–51. AAAI Press.

Saribatur, Z. G.; Schüller, P.; and Eiter, T. 2019. Abstraction for non-ground answer set programs. In *Proc. of the 16th European Conference on Logics in Artificial Intelligence (JELIA)*, LNCS. Springer. 576–592.