Unifying Justifications and Debugging for Answer-Set Programs

Carlos Viegas Damásio (*e-mail:* cd@fct.unl.pt)
NOVA LINCS, Universidade Nova de Lisboa, Portugal

João Moura∗ (*e-mail:* joaomoura@yahoo.com)
NOVA LINCS, Universidade Nova de Lisboa, Portugal

Anastasia Analyti (analyti@ics.forth.gr)
Institute of Computer Science, FORTH-ICS, Crete, Greece

submitted 29 April 2015; accepted 5 June 2015

Abstract

Recently, (Viegas Damásio et al. 2013) introduced a way to construct propositional formulae encoding provenance information for logic programs. From these formulae, justifications for a given interpretation are extracted but it does not explain why such interpretation is not an answer-set (debugging). Resorting to a meta-programming transformation for debugging logic programs, (Gebser et al. 2008) does the converse.

Here we unify these complementary approaches using meta-programming transformations. First, an answer-set program is constructed in order to generate every provenance propositional models for a program, both for well-founded and answer-set semantics, suggesting alternative repairs to bring about (or not) a given interpretation. In particular, we identify what changes must be made to a program in order for an interpretation to be an answer-set, thus providing the basis to relate provenance with debugging.

With this meta-programming method, one does not have the need to generate the provenance propositional formulas and thus obtain debugging and justification models directly from the transformed program. This enables computing provenance answer-sets in an easy way by using AS solvers. We show that the provenance approach generalizes the debugging one, since any error has a counterpart provenance but not the other way around. Because the method we propose is based on meta-programming, we extended an existing tool and developed a proof-of-concept built to help computing our models.

1 Introduction and Background

Theoretical results leading to tools for debugging answer-set programs have in the last few years been identified as a crucial prerequisite for a wider acceptance of answer-set programming (ASP). Tracing approaches (Bsusoniu et al. 2013) expose the user to the intricacies of reasoners (Brain et al. 2007) and declarative debugging approaches based in meta-programming techniques have instead been developed. But, one needs to understand why some interpretation is an answer-set (AS) –

∗ Under grant SFRH/BD/69006/2010 from Fundação para a Ciência e Tecnologia / Ministério do Ensino e da Ciência.

justifications – while understanding why some other interpretation is not an AS – **debugging**.

Example 1
Consider $\Pi = \{\mathbf{r_1} : a \leftarrow b. \quad \mathbf{r_2} : a \leftarrow not\ b.\}$. Besides knowing a is true in the single AS of Π, it is also be important to know that a is true because rules $\mathbf{r_1}$ and $\mathbf{r_2}$ are in Π, independently of what we can conclude about b. Another possible justification (among others) for a being true is that $\mathbf{r_2}$ is in Π and b has no support. Of course, if one intends a to be false then there is a *bug* in Π, with one justification being that rule $\mathbf{r_2}$ is unsatisfied.

In (Viegas Damásio et al. 2013), each literal can be associated with its *why not provenance* (WnP), i.e. a logical propositional formula explaining why a literal is true or false in an AS. In Example 1, formula $Why(a) = r_1 \wedge \neg not(b) \vee r_2 \wedge not(b) \vee \neg not(a)$ is obtained for literal a and its negation for *not* a. Clearly, $r_1 \wedge r_2 \models Why(a)$ is an intuitive justification for a. This provenance approach is capable of providing the corrections (adding or removing facts, and removing rules, e.g. $not(a) \wedge not(b) \wedge \neg r_2 \models \neg Why(a)$ in Ex. 1) that are necessary in order to bring about certain intended models.

Debugging ASP programs has been addressed in the literature by several authors, and the most effective approaches resort to meta-transformations to detect the diverse forms of anomalies in programs (Brain et al. 2007; Gebser et al. 2008; Oetsch et al. 2010; Polleres et al. 2013). Being fine grained, these are designed to pinpoint errors in LPs. Provenance formulas are inspired by a program transformation previously defined for declarative debugging of LPs (Pereira et al. 1993) and have been conjectured to be related (Viegas Damásio et al. 2013) with debugging.

The most important contributions we make are ***(1)*** bridging the gap between provenance models and LPs using meta-programming for ASP ***(2)*** unifying these two complementary approaches by mapping provenance models with debugging models (Gebser et al. 2008) ***(3)*** obtain justifications under the well-founded (WF) and AS semantics without calculating WnP formulae for a LP.

Structure Overview – Next we review relevant LP formalisms followed by debugging and provenance literature. In Section 2 we introduce a novel meta-programming transformation both for WF and AS semantics that is used to obtain models for WnP formulas of a given LP. We clarify which models are justifications, define them in terms of AS existence for the meta-program and discuss computational complexity. Section 3 provides a new mapping between our and the debugging transformations, showing that provenance captures debugging models but not the other way around. We end with a discussion, a comparison of these approaches with others in the literature and possible future work.

1.1 Logic Programming Formalisms

Normal logic programs (NLP) are sets of rules r of the following form:

$$r : A_1 \leftarrow A_2, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n . s.t., \ (n \geq m \geq 0)$$

where each A_i is a logical atom without function symbols. Let $Head(r) = \{A_1\}$, $Body^+(r) = \{A_2, \ldots, A_m\}$, $Body^-(r) = \{A_{m+1}, \ldots, A_n\}$, and $Body(r) = Body^+(r) \cup not\ Body^-(r)$. An (explicit) integrity constraint (IC) is a rule with the following form: $r : Head(r) \leftarrow Body(r), not\ Head(r)$. while its implicit form is r: $\leftarrow Body(r)$.

A program is definite (or positive) if it has no negated atoms. A disjunctive logic program (DLP) allows disjunction in the heads of rules. Without loss of generality, we assume that programs are grounded, and $Gr(\Pi)$ denotes the program obtained from Π by instantiating variables with constants occurring in Π. The Herbrand Base H_Π of a program Π is formed by the set of atoms occurring in it.

Given a set of literals J, let $not\ J = \{a \mid not\ a \in J\} \cup \{not\ a \mid a \in J$ s.t., for each b, $a \neq not\ b\}$. A two-valued interpretation is a subset of H_Π specifying the true atoms, and a partial interpretation is a subset of $H_\Pi \cup not\ H_\Pi$ (absent literals are undefined).

A two-valued interpretation I corresponds to the partial interpretation $I \cup not\ (H_\Pi \setminus I)$. The least model $least(\Pi)$ of a definite program Π is the least fixpoint of operator $T_\Pi(I) = \{Head(r) \mid r \in \Pi \wedge Body(r) \subseteq I\}$, where I is a two-valued interpretation.

The answer-sets of NLP Π are fixpoints of $\Gamma(I) = least(\Pi^I)$, where $\Pi^I = \{Head(r) \leftarrow Body^+(r) \mid r \in \Pi, Body^-(r) \cap I = \emptyset\}$.

The well-founded model (WFM) of Π is $T \cup not\ F$ where T and F are interpretations s.t., $T \cap F = \emptyset$, T is the least fixpoint $T = \Gamma(\Gamma(T)) = \Gamma^2(T)$ and $F = H_\Pi \setminus \Gamma(T)$.

1.2 Debugging

Debugging of LPs and in particular ASP has received important contributions over the last years. e.g., (Eiter et al. 2010) provided two approaches for explaining inconsistency, both of which characterize inconsistency in terms of bridge rules: by pointing out rules which need to be altered for restoring consistency, and by finding combinations of rules which cause inconsistency.

We are mostly interested in (Gebser et al. 2008) though, where a meta-programming technique for debugging ASPs is presented. Debugging queries are expressed by means of ASP programs, which allows restricting debugging information to relevant parts. The main question addressed is: "Why are interpretations expected to be answer-sets, not answer-sets of a given ASP program?" Thus it finds semantic errors in programs. The provided explanations are based on a scheme of errors relying in a characterization of AS semantics by (Lee 2005; Ferraris et al. 2007). However, these approaches do not answer the question of why a given (possibly unintended) interpretation is indeed an AS. In Theorem 2 of (Gebser et al. 2008), the four possible causes of errors dealt with by their debugging framework are:

Unsatisfied rules: If rule $r \in Gr(\Pi)$, with nonempty $Head(r)$, is unsatisfied by I, the logical implication expressed by r is false under I, i.e. $I \models Body(r)$ and $Head(r) \notin I$, and thus I is not a classical model of Π.

Violated integrity constraints: If an IC $c \in Gr(\Pi)$ is applicable under I, the fact that $Head(c) \neq \emptyset$ implies $I \not\models Body(c)$, and thus I cannot be an answer-set of Π.

Unsupported atoms: If $\{a\} \subseteq I$ is unsupported with respect to I, no rule in $Gr(\Pi)$ allows for deriving a, and thus I is not a minimal model of Π^I.

Unfounded loops: If a loop $\Gamma \subseteq I$ of Π is unfounded with respect to I, there is no acyclic derivation for the atoms in Π, and thus I is not a minimal model of Π^I.

They construct a meta-program (Fig. 1.2) from a given program Π and interpretation I that is capable of detecting the above errors via occurrences of the following error-indicating meta-atoms in its answer-sets: $unsatisfied(l_r)$ indicates that a rule $r \in Gr(\Pi)$ is unsatisfied by I; $violated(l_c)$ indicates that an IC $c \in Gr(\Pi)$ is violated under I; $unsupported(l_a)$ indicates that an atom $a \in I$ is unsupported; and $ufLoop(l_a)$ indicates that an atom a belongs to some unfounded loop $\Gamma \subseteq I$ of Π with respect to I.

$$
\begin{aligned}
\pi_{int} =\{&int(A) \leftarrow atom(A), not\ \overline{int}(A).\\
&\overline{int}(A) \leftarrow atom(A), not\ int(A).\}\\[1ex]
\pi_{sat} = \{&hasHead(R) \leftarrow head(R,\_).\\
&someHInI(R) \leftarrow head(R,A), int(A).\\
&violated(C) \leftarrow ap(C), not\ hasHead(C).hasHead(R),\\
&unsatisfied(R) \leftarrow ap(R), not\ someHInI(R).\}\\[1ex]
\pi_{supp} =\{&unsupported(A) \leftarrow int(A), not\ supported(A).\\
&supported(A) \leftarrow head(R,A), ap(R), not\ otherHInI(R,A).\\
&otherHInI(R,A1) \leftarrow head(R,A2), int(A2), head(R,A1), A1 \neq A2.\}\\[1ex]
\pi_{noas} =\{&noAnswerSet \leftarrow unsatisfied(\_). \qquad noAnswerSet \leftarrow violated(\_).\\
&noAnswerSet \leftarrow unsupported(\_). \qquad noAnswerSet \leftarrow ufLoop(\_).\\
&\leftarrow not\ noAnswerSet.\}\\[1ex]
\pi_{ap} =\{&bl(R) \leftarrow Body^+(R,A), \overline{int}((A). \qquad \%blocked\ rules\\
&bl(R) \leftarrow Body^-(R,A), int(A).\}\\
&ap(R) \leftarrow rule(R), not\ bl(R). \qquad \%applicable\ rules\\[1ex]
\pi_{ufloop} =\{&ufLoop(A) \leftarrow int(A), supported(A), not\ \overline{ufloop}(A).\\
&\overline{ufloop}(A) \leftarrow int(A), not\ ufLoop(A).\\
&someBInLoop(R) \leftarrow Body^+(R,A), ufLoop(A).\\
&someHOutLoop(R) \leftarrow head(R,A), \overline{ufloop}((A).\\
&dpcy(A1,A2) \leftarrow dpcy(A1,A3), dpcy(A3,A2).\\
&dpcy(A1,A2) \leftarrow head(R,A1), Body^+(R,A2), ap(R), ufLoop(A1), ufLoop(A2),\\
&\qquad not\ someHOutLoop(R).\\
&\leftarrow head(R,A), ufLoop(A), ap(R), not\ someHOutLoop(R),\\
&\qquad not\ someBInLoop(R).\\
&\leftarrow ufLoop(A1), ufLoop(A2), not\ dpcy(A1,A2).\}
\end{aligned}
$$

Fig. 1. Static Modules of Meta-Program $D(\Pi)$.

In (Gebser et al. 2008), the authors define the input meta-program $\pi_{in}(\Pi)$ from a ground DLP Π (note that we restrict our discussion to NLPs) as the following set of facts: $\boldsymbol{\pi_{in}(\Pi)} = \{atom(l_a) \leftarrow | a \in At(\Pi)\} \cup \{rule(l_r) \leftarrow | r \in \Pi\} \cup \{Head(l_r, l_a) \leftarrow | r \in \Pi, a \in Head(r)\} \cup \{Body^+(l_r, l_a) \leftarrow | r \in \Pi, a \in Body^+(r)\} \cup \{Body^-(l_r, l_a) \leftarrow | r \in \Pi, a \in Body^-(r)\}$. Program $\boldsymbol{\pi_{in}(\Pi)}$ consists of facts stating which rules and atoms occur in Π and, for each rule $r \in \Pi$, which atoms are contained in $Head(r)$, $Body^+(r)$, and $Body^-(r)$, respectively. Given $\pi_{in}(\Pi)$, the non-disjunctive **meta-program D(Π)** is defined as follows:

Let Π be a ground DLP. Then, the meta-program $D(\Pi)$ for Π consists of $\pi_{in}(\Pi)$

together with the modules of Fig. 1, i.e., $D(\Pi) = \pi_{in}(\Pi) \cup \pi_{int} \cup \pi_{ap} \cup \pi_{sat} \cup \pi_{supp} \cup \pi_{ufloop} \cup \pi_{noas}$.

Example 2
Consider program $\{\mathbf{r_1} : a \leftarrow b, c.\ \mathbf{r_2} : b \leftarrow d.\ \mathbf{r_3} : b \leftarrow not\ e.\ \mathbf{f_1} : c.\ \mathbf{f_2} : d.\}$, for which an intended AS is $I = \{b, c, d, e, f\}$. An explanation for I not being an AS is that r_1 is unsatisfied and e is unsupported. On the onther hand, (Gebser et al. 2008) cannot say why $\{a, b, c, d\}$ is an AS because a is true due to d being true due to e being false.

1.3 Provenance

In turn, (Viegas Damásio et al. 2013) presents a declarative logical approach to extract WnP information for logic programs. Using values of a freely generated Boolean algebra as annotation tags for atoms, they specify WnP for positive and NLPs under WF semantics, and relate it to abduction and calculation of prime implicants. The approach generalizes to ASP. These WnP formulae are used to determine provenance of literals true in a given model, and are shown in (Viegas Damásio et al. 2013) to extend the approaches of evidence graphs (Pemmasani et al. 2004) and off-line justifications (Pontelli et al. 2009). In the remaining of this section, assume that an LP Π over H_Π is given. **Why not provenance** is defined in (Viegas Damásio et al. 2013) and summarized bellow:

Let B_Π be the free Boolean algebra generated by propositional variables $H_\Pi \cup not(H_\Pi) \cup \{r_i | 1 \leq i \leq |\Pi|\}$, where for each rule of Π there is a unique and distinct rule identifier r_i. Elements of B_Π are the equivalence classes of propositional formulas under logical equivalence, and the partial ordering of B_Π is entailment: $[\phi] \preceq [\psi]$ iff $\phi \models \psi$. Thus B_Π is a lattice, with join and meet defined by $[\phi] \oplus [\psi] = [\phi \vee \psi]$, $[\phi] \otimes [\psi] = [\phi \wedge \psi]$, and let $[\phi] - [\psi] = [\phi \wedge \neg\psi]$. WnP is extracted with monotonic multivalued programs and a **WnP program $\mathcal{P}$ over H_Π** is defined as:

Let a WnP program $\mathcal{P}$ be formed by rules of the form $A \Leftarrow [J] \otimes B_1 \otimes \ldots \otimes B_m$ with $m \geq 0$, and where $[J] \in B_\Pi$ and $A, B_1, \ldots, B_m \in H_\Pi$. An interpretation I for $\mathcal{P}$ is a mapping $I : H_\Pi \rightarrow B_\Pi$. The set of all interpretations is a lattice with point-wise ordering. An interpretation I satisfies a rule $A \Leftarrow [J] \otimes B_1 \otimes \ldots \otimes B_m$ of program $\mathcal{P}$ iff $I(A) \succeq [J] \otimes I(B_1) \otimes \ldots \otimes I(B_m)$ iff $J \wedge I(B_1) \wedge \ldots \wedge I(B_m) \models I(A)$. Interpretation I is a model of $\mathcal{P}$ iff I satisfies all the rules of $\mathcal{P}$.

The monotonicity of $\mathcal{P}$ guarantees the existence of a least model $M_\mathcal{P}$ for it, and by mimicking the construction of a Gelfond-Lifschitz like operator, WnP for LPs under WF semantics is defined. The **rationale** for $\mathcal{P}$ is: If a program $\mathcal{P}$ has some fact A (resp. no fact for A), WnP formula for A is $[(r_i \wedge Why_i) \vee \ldots \vee (r_j \wedge Why_j) \vee A]$ (resp. $[(r_i \wedge Why_i) \vee \ldots \vee (r_j \wedge Why_j) \vee \neg not(A)]$). A justification for A is $[A]$ meaning there is a fact for A. Other justifications are obtained using a rule r_k and justifying why its body is true. The later case (denoted before by 'resp.') is better understood taking the justification for *not* A which has WnP formula $[\neg(r_i \wedge Why_i) \wedge \ldots \wedge \neg(r_j \wedge Why_j) \wedge not(A)]$, expressing that all bodies must be falsified and $[not(A)]$ holds.

Provenance program $\frac{\mathcal{P}}{\mathbf{I}}$ is constructed from Π and WnP interpretation I as follows:

- For the i^{th} rule $A \leftarrow B_1, \ldots, B_m, not\ C_1, \ldots, not\ C_n$ $(m+n \geq 1)$ in Π add provenance rule $A \Leftarrow [r_i \wedge \neg I(C_1) \wedge \ldots \neg I(C_n)] \otimes B_1 \otimes \ldots \otimes B_m$ to $\frac{\mathcal{P}}{I}$;
- $\forall A \in H_\Pi$, if $A \in \Pi$ (resp. $A \notin \Pi$) add $A \Leftarrow [A]$ (resp. $A \Leftarrow [\neg not(A)]$) to $\frac{\mathcal{P}}{I}$.

Operator $\mathfrak{G}_\Pi(I) = M_{\frac{\mathcal{P}}{I}}$ returns the least model of $\frac{\mathcal{P}}{I}$.

Operator $\mathfrak{G}_\Pi$ is anti-monotonic, and therefore $\mathfrak{G}^2_\Pi$ is monotonic having a least model $\mathfrak{T}_\Pi$, corresponding to provenance information for what is true in the WFM, while $\mathfrak{TU}_\Pi = \mathfrak{G}_\Pi(\mathfrak{T}_\Pi)$ contains the WnP of what is true or undefined in the WFM of Π. The following definitions capturing **Why-not provenance information under the well-founded semantics** are then provided:

Let $\mathfrak{T}_\Pi$ be the least model of $\mathfrak{G}^2_\Pi$, $\mathfrak{TU}_\Pi = \mathfrak{G}_\Pi(\mathfrak{T}_\Pi)$, and A be an atom. Let $Why_\Pi(A) = [\mathfrak{T}_\Pi(A)], Why_\Pi(not\ A) = [\neg\mathfrak{TU}_\Pi(A)]$, and $Why_\Pi(undef\ A) = [\neg\mathfrak{T}_\Pi(A) \wedge \mathfrak{TU}_\Pi(A)]$.

Theorem 1 (*Provenance Models for WF Semantics (Viegas Damásio et al. 2013)*)

Let $G \notin \Pi$ and $F \in \Pi$ (resp. $R \in \Pi$) be sets of facts (resp. rules). Literal $L \in WFM((\Pi \backslash (F \cup R)) \cup G)$ iff there is a conjunction of literals $C \models Why_\Pi(L)$ s.t., $RemoveFacts(C) \subseteq F, KeepFacts(C) \cap F = \emptyset, RemoveRules(C) \subseteq R, KeepRules(C) \cap R = \emptyset, MissingFacts(C) \subseteq G$, and $NoFacts(C) \cap G = \emptyset$ where $NoFacts(C)$ (resp. $MissingFacts(C)$) are facts that cannot (resp. must) be added:

$\mathbf{KeepFacts(C)} = \{A. \mid A \in F\}$ $\quad$ $\mathbf{RemoveFacts(C)} = \{A. \mid \neg A \in F\}$
$\mathbf{KeepRules(C)} = \{r_i : A \leftarrow Body \mid r_i \in R \text{ and } r_i \in \Pi\}$
$\mathbf{RemoveRules(C)} = \{r_i : A \leftarrow Body \mid \neg r_i \in R \text{ and } r_i \in \Pi\}$
$\mathbf{NoFacts(C)} = \{A. \mid not(A) \in G\}$ $\quad$ $\mathbf{MissingFacts(C)} = \{A. \mid \neg not(A) \in G\}$

Example 3 (*From (Viegas Damásio et al. 2013)*)
Consider again program Π in Ex. 2. Its WnP information is:

$Why(a) = [r_1 \wedge c \wedge ((r_2 \wedge d) \vee (r_3 \wedge not(e)) \vee \neg not(b)) \vee \neg not(a)]$
$Why(not\ a) = [not(a) \wedge (\neg r_1 \vee \neg c \vee (not(b) \wedge (\neg r_2 \vee \neg d) \wedge (\neg r_3 \vee \neg not(e))))]$
$Why(b) = [(r_2 \wedge d) \vee (r_3 \wedge not(e)) \vee \neg not(b)]$
$Why(not\ b) = [not(b) \wedge (\neg r_2 \vee \neg d) \wedge (\neg r_3 \vee \neg not(e))]$
$Why(c) = [c]$ $\quad$ $Why(not\ c) = [\neg c]$ $\quad$ $Why(d) = [d]$
$Why(not\ d) = [\neg d]$ $\quad$ $Why(e) = [\neg not(e)]$ $\quad$ $Why(not\ e) = [not(e)]$

The provenance for a being false, and all other atoms true is derived from the models of $Why(not\ a) \wedge Why(b) \wedge Why(c) \wedge Why(d) \wedge Why(e) = not(a) \wedge \neg r_1 \wedge (r_2 \vee \neg not(b)) \wedge c \wedge d \wedge \neg not(e)$, thus a fact for a must be absent, we have to remove rule r_1, keep rule r_2 or add fact b, keep facts c and d, and add fact e. (Gebser et al. 2008) detects that rule r_1 is unsatisfied and e is unsupported but it does not determine provenance for a. One way to make a true is to simply add a fact for it; alternatively r_1 must be kept in Π as well as facts c and b. This is achieved by: keeping r_2 and d; keeping r_3 and not adding e; adding b.

One obtains AS provenance from the WFM provence basically by forcing all atoms to be either positive or negative, i.e., non-undefined, and using the provenance determined for the WF semantics (see examples in Section 2).

Justifications are defined as: Given a logic program Π and a literal l, a justification J for l in Π as an implicant of the why provenance formula $Why_\Pi(l)$, i.e. a conjunction of literals s.t., $J \models Why_\Pi(l)$. The implicant is *prime* if there is no other implicant J' of $Why_\Pi(l)$ with less literals.

Note that it has been proved in (Viegas Damásio et al. 2013) that evidence graphs (Pemmasani et al. 2004) and off-line justifications (Pontelli et al. 2009) models can all be captured by WnP implicants, but some of our justifications cannot be mapped by them. Also related to provenance are causal chains (Cabalar and Fandiño 2013) where a multi-valued semantics for NLPs whose truth values form a lattice of causal chains is provided. A causal chain is a concatenation of rule labels reflecting some sequence of rule applications.

2 Contributions: Meta-Transformation for Provenance Formulae

We define here a novel program transformation, capable of obtaining all models of WnP formulae, composed of two parts: **(1)** a set of common modules in Fig. 2, shared by specific transformations for WF and AS semantics; **(2)** WF specific modules in Fig. 3. Its vocabulary is based in (Gebser et al. 2008) to ease their subsequent combining. We only deal with the non-disjunctive case and ICs must be in their explicit form.

$$
\begin{aligned}
\pi_{fact} = \{ & fact(X) \leftarrow rule(R), head(R,X), not\ hasBody(R). \\
& hasBody(R) \leftarrow rule(R), bodyP(R,A). \\
& hasBody(R) \leftarrow rule(R), bodyN(R,A).\} \\
\pi_{Rules} = \{ & keepRule(X) \leftarrow rule(X), not\ removeRule(X). \\
& removeRule(X) \leftarrow rule(X), not\ keepRule(X).\} \\
\pi_{Facts} = \{ & missingFact(X) \leftarrow atom(X), not\ fact(X), not\ noFact(X). \\
& noFact(X) \leftarrow atom(X), not\ fact(X), not\ missingFact(X).\}
\end{aligned}
$$

Fig. 2. Common Provenance Modules π_{common}

Module $\pi_{fact} \in \pi_{common}$ defines facts as rules in the program having empty bodies. Module π_{fact} assumes that module $\pi_{in} \in D(\Pi)$ (Fig. 1.2) will also be applied, since it depends on all facts defined in π_{in}. Modules π_{Rules} and π_{Facts} are the generators for propositional variables used in the provenance formulae in the vocabulary of Theorem 1. Note that in π_{Rules} the provenance propositional variables for facts H_Π are captured by $keepRule/1$ since, for generality purposes, $rule/1$ represents both rule and fact identifiers.

$$
\begin{aligned}
\pi_{ttu} = \{ & \leftarrow atom(A), t(A), not\ tu(A).\\
& \leftarrow head(R,H), not\ ap(R,t), not\ ap(R,tu), not\ undef(H), removeRule(R).\\
& t(H) \leftarrow head(R,H), keepRule(R), ap(R,t), not\ undef(H).\\
& t(H) \leftarrow atom(H), missingFact(H), not\ fact(H), not\ undef(H).\\
& tu(H) \leftarrow head(R,H), keepRule(R), ap(R,tu).\\
& tu(H) \leftarrow atom(H), missingFact(H), not\ fact(H).\\
& tu(H) \leftarrow atom(H), undef(H).\}
\end{aligned}
$$

$$
\begin{aligned}
\pi_{apttu}(\Pi) = \{ & ap(r_i,t) \leftarrow t(B_1), \ldots t(B_m), not\ tu(C_1), \ldots, not\ tu(C_n).,\\
& ap(r_i,tu) \leftarrow tu(B_1), \ldots tu(B_m), not\ t(C_1), \ldots, not\ t(C_n).\\
& \quad \mid A \leftarrow B_1, \ldots B_m, not\ C_1, \ldots, not\ C_n \in \Pi \text{ and identified by } r_i.\}
\end{aligned}
$$

Fig. 3. Meta transformation π_{wfs} modules

2.1 Provenance for the WF Semantics

A provenance program under the WF semantics is captured by π_{wfs} combined with debugging modules π_{common} and π_{in}. Module π_{ttu} encodes the Γ^2 operator for the program subject to changes defined by pairs *keepRule*/*removeRule* and *missingFact*/*noFact*, where predicate $t/1$ represents what is true (the outer Γ), and $tu/1$ what is true or undefined (the inner Γ). The constraint discards models where assignments are contradictory, ensuring that $t(A) \Rightarrow tu(A)$ for every atom A. The module also uses an extra meta-predicate $undef/1$ that allows to make an atom undefined, a new kind of change not captured by the original provenance model for WF semantics that is included for the sake of completeness. Module π_{apttu} determines when a rule is applicable in the outer ($ap(R,t)$), and inner steps ($ap(R,tu)$), and generalizes module π_{ap} of (Gebser et al. 2008).

Lemma 1
Given a logic program Π and a propositional model M of formula $Why_\Pi(L)$ for a literal L, then there is an AS M' of $W(\Pi) = \pi_{in}(\Pi) \cup \pi_{common} \cup \pi_{wfs}(\Pi)$ s.t.:

- If $A \in H_\Pi$ s.t., fact $A \in \Pi$ and $A \in M$ then $keepRule(r_{A\leftarrow}) \in M'$ and $removeRule(r_{A\leftarrow}) \notin M'$
- If $A \in H_\Pi$ s.t., fact $A \in \Pi$ and $A \notin M$ then $keepRule(r_{A\leftarrow}) \notin M'$ and $removeRule(r_{A\leftarrow}) \in M'$
- If $not(A) \in H_\Pi$ s.t., no fact A in Π and $not(A) \notin M$ then $missingFact(A) \in M'$ and $noFact(A) \notin M'$
- If $not(A) \in H_\Pi$ s.t., no fact A in Π and $not(A) \in M$ then $missingFact(A) \notin M'$ and $noFact(A) \in M'$
- If $r_i \in M$ then $keepRule(r_i) \in M'$ and $removeRule(r_i) \notin M'$
- If $r_i \notin M$ then $keepRule(r_i) \notin M'$ and $removeRule(r_i) \in M'$

For the converse direction extra answer-sets of $W(\Pi)$ may be generated. When we fix the changes to Π all partial stable models (PSM) of the changed program are obtained, i.e. all fixpoints of Γ^2, instead of solely the least fixpoint of Γ^2. These models can be filtered out by guaranteeing minimality of the model. This is captured in the following Lemmata:

Lemma 2
Let $M' \in AS(W(\Pi) = \pi_{in}(\Pi) \cup \pi_{common} \cup \pi_{wfs}(\Pi))$ and $Model(M') = \{A \mid t(A) \in M'\} \cup \{not\ A \mid tu(A) \notin M'\}$. Construct program Π' by deleting from Π every rule identified by r_i s.t., $removeRule(r_i) \in M'$, and adding a fact A in Π' for every $missingFact(r_{A\leftarrow}) \in M'$. Then, $Model(M')$ is a PSM of Π'. Conversely, if Π' is a program obtained deleting rules or adding facts, then every PSM of Π' has a corresponding AS in $W(\Pi)$.

Lemma 3
Let M' be an AS of $W(\Pi) = \pi_{in}(\Pi) \cup \pi_{common} \cup \pi_{wfs}(\Pi)$, s.t., there is no AS M'' of $W(\Pi)$ with $Model(M'') \subsetneq Model(M')$. Let M be the model obtained from M' by reverting transformation in Lemma 1. Then, M is a model of $Why_\Pi(L)$ for each $L \in M'$.

Example 4
Recall now Ex. 2, to which we apply transformation $W(\Pi)$ where $\pi_{in}(\Pi) = \{head(r_1, a).\ bodyP(r_1, b).\ bodyP(r_1, c).\ head(r_2, b).\ bodyP(r_2, d).\ head(r_3, b).\ bodyN(r_3, e).\ head(f_1, c).\ head(f_2, d).\}$. $W(\Pi)$ has 256 answer-sets corresponding to all possible changes to Π by removing or keeping rules, and adding or not facts. Of these answer-sets, 6 correspond to a being false and all other atoms true, and are in exact correspondence with the propositional models of formula $not(a) \wedge \neg r_1 \wedge (r_2 \vee \neg not(b)) \wedge c \wedge d \wedge \neg not(e)$ obtained in Ex. 3. All answer-sets below contain[1] $\{noFact(a), removeRule(r_1), keepRule(f_1; f_2), missingFact(e)\}$, corresponding to conjuncts $not(a), \neg r_1, c, d, \neg not(e)$ of the previous formula and are filtered for readability:

$$\{keepRule(r_2; r_3), missingFact(b)\}$$
$$\{keepRule(r_2; r_3), noFact(b)\}$$
$$\{keepRule(r_2), removeRule(r_3), missingFact(b)\}$$
$$\{keepRule(r_2), removeRule(r_3), noFact(b)\}$$
$$\{removeRule(r_2), keepRule(r_3), missingFact(b)\}$$
$$\{removeRule(r_2; r_3), missingFact(b)\}$$

There are 151 possible AS explaining why a is true, corresponding to the 151 models of $Why_\Pi(a)$.

2.2 Provenance for the Answer-Set Semantics

Forbidding undefined atoms in the model and disallowing models where $t/1$ does not occur when $tu/1$ occurs (Fig. 4), adapts the WF transformation presented before to the AS semantics.

Theorem 2 follows from the above Lemmata 1, 2 and 3 for the WF semantics, but first we need an auxiliary notion defining what is the WnP of an intended AS:

[1] we denote a set of predicates $\{a(X), ..., a(Y)\}$ as $a(X; ...; Y)$

$$\pi_{as}(\Pi) = \pi_{common} \cup \pi_{wfs}(\Pi) \cup \{\leftarrow atom(A), tu(A), not\ t(A).\ \leftarrow atom(A), undef(A).\}$$

Fig. 4. Meta-transformation π_{as}

Definition 1 (*Why not provenance for an interpretation*)
Let Π be a logic program and I an interpretation. The **AS WnP for *I*** is:

$$AnsWhy^I_\Pi = \bigwedge_{\forall A \in I} AnsWhy_\Pi(A) \bigwedge_{\forall A \notin I} AnsWhy_\Pi(not\ A) \bigwedge_{\forall A \notin I} \{\neg r | A \in Head(r)\}$$

Intuitively, the WnP for an interpretation I is the intersection of the WnP for its positive (and negative) atoms. We then select WnP formulae containing $\neg r$ for every $r \in \Pi$ s.t., an atom $A \notin I$ belongs to $Head(r)$ which effectively avoids considering rules giving support to unintended atoms, and thus providing unnecessary justifications. This forms a restricted class, containing interesting WnP formulas, for which the following applies:

Theorem 2 (*WnP models for the AS semantics*)
Given a logic program Π, and an interpretation I, then the answer-sets of transformed program $S(\Pi) = \pi_{in}(\Pi) \cup \pi_{as}(\Pi)$ s.t., $Model(M) = I$ are in 1:1 correspondence with the models of $AnsWhy^I_\Pi$.

These models are complete in the sense they provide all possible justifications for an AS or all explanations for why an interpretation is not a model. These can then be minimized according to whatever criteria one might have, e.g., subset minimality, minimal changes to the program, disallowing or preferring certain repair operations over others etc., which can be captured by optimization constraints supported by the major ASP solvers.

Consider now again the program in Example 2: $\{a \leftarrow c, not\ b.\ b \leftarrow not\ a.\ d \leftarrow not\ c, not\ d.\ c \leftarrow not\ e.\ e \leftarrow f.\ f \leftarrow e.\}$. This program has answer-sets $A_1 : \{a, c\}$ and $A_2 : \{b, c\}$. Below are some of the 144 WnP models for A_1 from which we select the ones presenting intuitive explanations (model selection is clarified next) from all of which the following literals are omitted: $F = \{keepRule(r2; r3; r5; r6), noFact(b; d; e; f), t(a; c)\}$: **(1)** $\{removeRule(r1), keepRule(r4), missingFact(a; c)\}$ **(2)** $\{removeRule(r1; r4), missingFact(a; c)\}$ **(3)** $\{removeRule(r1), keepRule(r4), missingFact(a), noFact(c)\}$ **(4)** $\{keepRule(r1), removeRule(r4), noFact(a), missingFact(c)\}$ **(5)** $\{keepRule(r1; r4), missingFact(a; c)\}$ **(6)** $\{keepRule(r1; r4), noFact(a), missingFact(c)\}$ **(7)** $\{keepRule(r1; r4), missingFact(a), noFact(c)\}$ **(8)** $\{keepRule(r1), removeRule(r4), missingFact(a; c)\}$ **(9)** $\{keepRule(r1; r4), noFact(a; c)\}$

3 Contributions: Mapping Provenance with Debugging

As shown before, our meta-transformation produces a model for each WnP model and some can be aligned with debugging models calculated by (Gebser et al. 2008). These approaches complement each other: we produce provenance models for existing answer-sets, while the debugging approach is capable of obtaining more specific

results regarding the non-existence of answer-sets, namely in the presence of unfounded loops. So, we need to impose equivalence between predicates $int/1$ and $t/1$ (see Fig. 5) and thus introduce of module π_{map}. The resulting models consist of two parts, one stating what is the problem with the interpretation at hand (corresponding to the debugging part) and the other offering a justification for why this interpretation is a model of the program (corresponding to the provenance part).

$$
\begin{aligned}
\pi_{t-int} = \{ \quad & \leftarrow atom(A), int(A), not\ t(A). \\
& \leftarrow atom(A), \overline{int}(A), t(A).\} \\
\\
\pi_{ics} = \{ \quad & \leftarrow rule(R), violated(R), keepRule(R).\} \\
\\
\pi_{prune} = \{ \quad & \leftarrow rule(R), removeRule(R), not\ ap(R). \\
& \leftarrow atom(A), missingFact(A), supported(A), not\ ufLoop(A). \\
& \leftarrow atom(A), noFact(A), supported(A), ufLoop(A).\}
\end{aligned}
$$

Fig. 5. Transformation $\pi_{map} = \pi_{t-int} \cup \pi_{ics} \cup \pi_{prune}$

Module π_{t-int} ensures that atoms int and t are mapped which effectively maps *provenance* and *debugging* at the interpretation level, while π_{ics} guarantees that violated ICs are corrected by removing them. The combined program is $J(\Pi) = \pi_{int} \cup \pi_{sat} \cup \pi_{supp} \cup \pi_{ufLoop} \cup \pi_{as}(\Pi) \cup \pi_{map} \cup D'(\Pi)$, where in order to determine provenance for a given AS, $D'(\Pi)$ is obtained from $D(\Pi)$ by substituting π_{ap} with π_{apttu} and removing π_{noas}.

Intuitively, an interpretation is guessed (represented by $int/1$), and one then forces the correspondence of $t/1$ with $int/1$. The **repaired program** (removing rules or adding missing facts) is guessed, and generates the extension of $t/1$, and it is always possible to trivially repair a program and obtain any desired interpretation by removing all rules and adding all missing facts. We now look at error-indicating predicates to detect problems with Π.

Theorem 3
Let M be an AS of $D(\Pi)$. Then, there is an AS M' of meta-program $J(\Pi)$ s.t., $M \setminus (\{noAnswerSet\} \cup \{ap(r_i), bl(r_i) \mid r_i \text{ is a rule identifier}\}) \subseteq M'$.

So, we are able to detect every error pointed out by error-indicating predicates of (Gebser et al. 2008). There is however a subtle difference: we prune debugging answer-sets which are not supported by the repaired program. Their exact relationship is captured next:

Theorem 4 (Mapping)
Let M' be an answer-set of $J(\Pi)$. Then,

- If $unsatisfied(r_i)$ or $violated(r_i) \in M'$ then $removeRule(r_i) \in M'$;
- If $unsupported(r_i) \in M'$ then $missingFact(r_i) \in M'$;
- If $ufLoop(a_1..a_n) \in M'$ then $\exists i \in [1, \ldots, n]$ s.t., $missingFact(a_i) \in M'$. Also, $\exists M'' \in AS(J(\Pi))$ s.t. $ufLoop(a_1..a_n) \cup missingFact(a_1..a_n) \in M''$.

However, some provenance answer-sets may be considered redundant (even though correct) but module π_{prune} in Fig. 5 can be used to prune them. It disallows removing blocked rules ($bl/1$), adding facts which are not in unfounded loops but are already supported, and forces a $missingFact$ to be added to every atom in detected unfounded loops.

Example 5
Take again Ex. 1 in (Viegas Damásio et al. 2013) and include relevant modules of transformation D. We show next a sample of its answer-sets, having $F = \{keepRule(r1;\ r2;\ r3;\ r4; r6),\ unsupported(a; b),\ missingFact(a; b),\ noFact(c; e)\}$ in common.

$F \cup \{removeRule(r5), unsupported(d; f), unsatisfied(r5), missingFact(d; f)\}$
$F \cup \{removeRule(r5), unsatisfied(r5), supported(c; e), noFact(d), unsupported(f), missingFact(f)\}$
$F \cup \{keepRule(r5), supported(c), unsupported(d), missingFact(d), noFact(f)\}$
$F \cup \{keepRule(r5), supported(c), noFact(d; f)\}$

4 Conclusions and Future Work

We provide a transformation to compute WnP models under the WF and AS semantics by computing the answer-sets of meta-programs that capture the original programs and include some necessary extra atoms. We do this in a modular way, preserving compatibility with the previous work of (Viegas Damásio et al. 2013) and computing the models directly without first obtaining the provenance formulas for certain interpretations. This enables computing provenance answer-sets in an easy way by using AS solvers. Having this, we align provenance and debugging answer-sets in a unified transformation and show that the provenance approach generalizes the debugging one, since any error has a counterpart provenance but not the other way around. Since the proposed method is based on meta-programming, we extended an existing tool (Gebser et al. 2007) and developed a proof-of-concept (*http://cptkirk.sourceforge.net*) built solely to allow computing our models.

Our mapping allows generating answer-sets capturing errors and justifications for (intended) models. As expected, they are exponential. One direction to explore is to obtain prime implicant by optimizing these models using reification and then subset inclusion preference ordering (Gebser et al. 2007; Gebser et al. 2011) via a saturation technique (Eiter and Gottlob 1995). Note that deciding if an AS is optimal for a DLP is a Π_2^p-complete problem. Alternative *offline justifications* (Pontelli et al. 2009) (which are also exponential) can be extracted from models of $J(\Pi)$ by adding extra constraints to the transformed program guaranteeing: only one rule is kept for true atoms (providing support); literals assumed false have all rules removed (which are undefined in the WFM); false literals have to keep all their rules; the dependency graph is acyclic; The major difference to Pontelli's approach is that we provide justifications for the full model from which we may obtain their justifications, but our approach subsumes it since we are capable of findng more justifications as well as errors in the program.

References

Brain, M., Gebser, M., Pührer, J., Schaub, T., Tompits, H., and Woltran, S. 2007. Debugging asp programs by means of asp. In *LPNMR 2007* (2007-06-06), C. Baral, G. Brewka, and J. S. Schlipf, Eds. Lecture Notes in Computer Science, vol. 4483. Springer, 31–43.

Bsusoniu, P.-A., Oetsch, J., Pührer, J., Skočovsky, P., and Tompits, H. 2013. Sealion: An eclipse-based ide for answer-set programming with advanced debugging support. *Theory and Practice of Logic Programming 13*, 657–673.

Cabalar, P. and Fandiño, J. 2013. An algebra of causal chains. In *Logic Programming and Nonmonotonic Reasoning*, P. Cabalar and T. Son, Eds. Lecture Notes in Computer Science, vol. 8148. Springer Berlin Heidelberg, 530–542.

Eiter, T., Fink, M., Schüller, P., and Weinzierl, A. 2010. Finding explanations of inconsistency in multi-context systems. *KR 10*, 329–339.

Eiter, T. and Gottlob, G. 1995. On the computational cost of disjunctive logic programming: Propositional case.

Ferraris, P., Lee, J., and Lifschitz, V. 2007. A new perspective on stable models. In *Proceedings of the 20th international joint conference on Artifical intelligence*. IJCAI'07. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 372–379.

Gebser, M., Kaminski, R., and Schaub, T. 2011. Complex optimization in answer set programming. *Theory and Practice of Logic Programming 11*, 821–839.

Gebser, M., Kaufmann, B., Neumann, A., and Schaub, T. 2007. clasp: A conflict-driven answer set solver. In *Logic Programming and Nonmonotonic Reasoning*. Springer Berlin Heidelberg, 260–265.

Gebser, M., Puehrer, J., Schaub, T., and Tompits, H. 2008. A meta-programming technique for debugging answer-set programs. In *AAAI-08/IAAI-08 Proceedings*, D. Fox and C. P. Gomes, Eds. 448–453.

Gebser, M., Pührer, J., Schaub, T., Tompits, H., and Woltran, S. 2007. spock: A Debugging Support Tool for Logic Programs under the Answer-Set Semantics. In *Proceedings of the 21st Workshop on (Constraint) Logic Programming, (WLP'07), Würzburg, Germany*, D. Seipel, M. Hanus, A. Wolf, and J. Baumeister, Eds. Technical Report 434, Bayerische Julius-Maximilians-Universität Würzburg, Institut für Informatik, 258–261.

Lee, J. 2005. A model-theoretic counterpart of loop formulas. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*. Professional Book Center, 503–508.

Oetsch, J., Pührer, J., and Tompits, H. 2010. Catching the ouroboros: On debugging non-ground answer-set programs. *Theory Pract. Log. Program. 10,* 4-6 (July), 513–529.

Pemmasani, G., Guo, H.-F., Dong, Y., Ramakrishnan, C., and Ramakrishnan, I. 2004. Online justification for tabled logic programs. In *Functional and Logic Programming*, Y. Kameyama and P. Stuckey, Eds. Lecture Notes in Computer Science, vol. 2998. Springer Berlin Heidelberg, 24–38.

Pereira, L. M., Damásio, C. V., and Alferes, J. J. 1993. Diagnosis and debugging as contradiction removal. In *Proceedings of the 2nd International Workshop on Logic Programming and Non-monotonic Reasoning*. MIT Press, 316–330.

Polleres, A., Frhstck, M., Schenner, G., and Friedrich, G. 2013. Debugging non-ground asp programs with choice rules, cardinality and weight constraints. In *Logic Programming and Nonmonotonic Reasoning*, P. Cabalar and T. Son, Eds. Lecture Notes in Computer Science, vol. 8148. Springer Berlin Heidelberg, 452–464.

Pontelli, E., Son, T. C., and El-Khatib, O. 2009. Justifications for logic programs under answer set semantics. *TPLP 9,* 1, 1–56.

Viegas Damásio, C., Analyti, A., and Antoniou, G. 2013. Justifications for logic programming. In *Logic Programming and Nonmonotonic Reasoning*, P. Cabalar and T. C. Son, Eds. Lecture Notes in Computer Science, vol. 8148. Springer Berlin Heidelberg, 530–542.