

Justifications for Logic Programs under Answer Set Semantics

Enrico Pontelli, Tran Cao Son, and Omar Elkhatib

Department of Computer Science
New Mexico State University
epontell|tson|okhatib@cs.nmsu.edu

submitted 24 May 2007; revised 22 March 2008, 19 September 2008; accepted 27 November 2008

Abstract

The paper introduces the notion of *off-line justification* for Answer Set Programming (ASP). Justifications provide a graph-based explanation of the truth value of an atom w.r.t. a given answer set. The paper extends also this notion to provide justification of atoms *during* the computation of an answer set (*on-line justification*), and presents an integration of on-line justifications within the computation model of SMOBELS. Off-line and on-line justifications provide useful tools to enhance understanding of ASP, and they offer a basic data structure to support methodologies and tools for *debugging* answer set programs. A preliminary implementation has been developed in ASP – PROLOG.

KEYWORDS: answer set programming, justification, offline justification, online justification

1 Introduction

Answer set programming (ASP) is a programming paradigm (Niemelä 1999; Marek and Truszczyński 1999; Lifschitz 2002) based on logic programming under answer set semantics (Gelfond and Lifschitz 1988). ASP is a *highly declarative* paradigm. In order to solve a problem P , we specify it as a logic program $\pi(P)$, whose answer sets correspond one-to-one to solutions of P , and can be computed using an answer set solver. ASP is also attractive because of its numerous building block results (see, e.g., (Baral 2003)). This can be seen in the following example.

Example 1

Consider the problem of computing the Hamiltonian cycles of a graph. The graph can be encoded as a collection of facts, e.g.,

vertex(a).	vertex(b).	vertex(c).	vertex(d).
edge(a,b).	edge(a,c).	edge(b,d).	edge(b,c).
edge(c,d).	edge(d,a).		

A program contains rules, in the form of Horn clauses; in our case:

```

%% Select an edge
in(U,V)      :- edge(U,V), not nin(U,V).
nin(U,V)     :- edge(U,V), not in(U,V).
%% Traverse each node only once
false       :- vertex(U), vertex(V), vertex(W),
              V ≠ W, in(U,V), in(U,W).
false       :- vertex(U), vertex(V), vertex(W),
              U ≠ V, in(U,W), in(V,W).
%% Reachability of nodes
reachable(U) :- vertex(U), in(a,U).
reachable(V) :- vertex(V), vertex(U), reachable(U), in(U,V).
%% Each vertex reachable from a
false       :- vertex(U), U ≠ a, not reachable(U).

```

It can be shown that every answer set of the program consisting of the rules representing the graph and the above rules corresponds to an Hamiltonian cycle of the graph and vice versa. Furthermore, the program has no answer set if and only if the graph does not have an Hamiltonian cycle. \square

The popularity of ASP has grown significantly over the years, finding innovative and highly declarative applications in a variety of domains, such as intelligent agents (Baral 2003; Balduccini et al. 2006), planning (Lifschitz 1999), software modeling and verification (Heljanko and Niemelä 2003), complex systems diagnosis (Balduccini and Gelfond 2003), and phylogenetic inference (Erdem et al. 2006).

The growing popularity of ASP, especially in domains like non-monotonic and commonsense reasoning, has been supported by the development of excellent inference engines (Anger et al. 2005; Eiter et al. 1998; Gebser et al. 2007; Giunchiglia et al. 2004; Lin and Zhao 2002; Simons et al. 2002). On the other hand, a source of difficulties in learning to use ASP lies in the lack of *methodologies* and *tools* which can assist users in understanding a program’s behavior and debugging it. The highly declarative nature of the ASP framework and the “hand-off” execution style of ASP leave a programmer with little information that helps in explaining the behavior of the programs, except for the program itself. For example, the additional information that can be gained by exploring the intermediate state of an execution (e.g., value of variables) of an imperative program using a debugger does not have any equivalent in the context of ASP. This situation is especially difficult when the program execution produces unexpected outcomes, e.g., incorrect or missing answer sets. In this sense, this paper shares the spirit of other attempts in developing tools and methodologies for understanding and debugging of ASP programs,¹ as in (Brain et al. 2007b; Brain et al. 2007a; El-Khatib et al. 2005; Perri et al. 2007).

Although the traditional language of logic programming under answer set semantics, e.g., referred to as AnsProlog in (Baral 2003) or A-Prolog (Gelfond and Leone

¹ Abusing the notation, we often refer to a logic program under the answer set semantics as an “ASP program” whenever it is clear from the context what it refers to.

2002), is syntactically close to Prolog, the execution model and the semantics are sufficiently different to make debugging techniques developed for Prolog impractical. For example, the traditional *trace-based* debuggers (Roychoudhury et al. 2000) (e.g., Prolog four-port debuggers), used to trace the entire proof search tree (paired with execution control mechanisms, like spy points and step execution), are cumbersome in ASP, since:

- Trace-based debuggers provide the entire search sequence, including the failed paths, which might be irrelevant in understanding how specific elements are introduced in an answer set.
- The process of computing answer sets is bottom-up, and the determination of the truth value of one atom is intermixed with the computation of other atoms; a direct tracing makes it hard to focus on what is relevant to one particular atom. This is illustrated in the following example.

Example 2

Consider the following simple program.

$$\begin{array}{ll} \mathbf{s} & :- \mathbf{r}. & \mathbf{s} & :- \mathbf{t}. \\ \mathbf{r} & :- \mathbf{a}. & \mathbf{t}. & \end{array}$$

The program P has a unique answer set, $M = \{\mathbf{s}, \mathbf{t}\}$. In particular, $\mathbf{t} \in M$, since \mathbf{t} appears as a fact in the program, and $\mathbf{s} \in M$ because of the rule $\mathbf{s} :- \mathbf{t}$ and $\mathbf{t} \in M$. In this process, there is no need to expose the processing of the rule $\mathbf{s} :- \mathbf{r}$ to the user, since $\mathbf{r} \notin M$. □

- Tracing repeats previously performed executions, degrading debugging performance and confusing the programmer.

We address these issues by elaborating the concept of *off-line justification* for ASP. This notion is an evolution of the concept of *justification*, proposed to justify truth values in tabled Prolog (Roychoudhury et al. 2000; Pemmasani et al. 2004). Intuitively, an off-line justification of an atom w.r.t. an answer set is a graph encoding the reasons for the atom’s truth value. This notion can be used to explain the presence or absence of an atom in an answer set, and provides the basis for building a *justifier* for answer set solvers. In this paper, we develop this concept and investigate its properties.

The notion of off-line justification is helpful when investigating the content of one (or more) answer sets. When the program does not have answer sets, a different type of justification is needed. This leads us to the notion of *on-line justification*, which provides justifications with respect to a *partial* and/or (sometimes) *inconsistent* interpretation. An on-line justification is *dynamic*, in that it can be obtained at any step of the answer set computation, provided that the computation process follows certain strategies. The intuition is to allow the programmer to interrupt the computation (e.g., at the occurrence of certain events, such as assignment of a truth value to a given atom) and to use the on-line justification to explore the motivations behind the content of the partial interpretation (e.g., why a given atom is receiving conflicting truth values). We describe a *generic* model of on-line justification and

a version specialized to the execution model of SMOBELS (Simons et al. 2002). The latter has been implemented in ASP – PROLOG (Elkhatib et al. 2004).

Justifications are offered as first-class citizens of a Prolog system, enabling the programmer to use Prolog programs to reason about ASP computations. Debugging is one of the possible uses of the notion of justification developed in this paper.

2 Background

In this paper, we focus on a logic programming language with negation as failure—e.g., the language of SMOBELS without weight constraints and choice rules (Simons et al. 2002).

2.1 Logic Programming Language

Each program P is associated with a signature $\Sigma_P = \langle \mathcal{F}, \Pi, \mathcal{V} \rangle$, where

- \mathcal{F} is a finite set of *constants*,
- \mathcal{V} is a set of *variables*, and
- Π is a finite set of *predicate* symbols.

In particular, we assume that \top (stands for *true*) and \perp (stands for *false*) are zero-ary predicates in Π . A *term* is a constant of \mathcal{F} or a variable of \mathcal{V} . An *atom* is of the form $p(t_1, \dots, t_n)$, where $p \in \Pi$, and t_1, \dots, t_n are terms. In particular, a term (atom) is said to be *ground* if there are no occurrences of elements of \mathcal{V} in it.

A *literal* is either an atom (*Positive Literal*) or a formula of the form *not* a , where a is an atom (*NAF Literal*). In what follows, we will identify with \mathcal{L} the set of all ground literals.

A *rule*, r , is of the form

$$h \text{ :- } b_1, \dots, b_n. \quad (1)$$

($n \geq 0$) where h is an atom and $\{b_1, \dots, b_n\} \subseteq \mathcal{L}$. The atom h is referred to as the *head* of the rule, while the set of literals $\{b_1, \dots, b_n\}$ represents the *body* of the rule. Given a rule r , we denote h with $head(r)$ and we use $body(r)$ to denote the set $\{b_1, \dots, b_n\}$. We also denote with $pos(r) = body(r) \cap \mathcal{A}$ —i.e., all the elements of the body that are not negated—and with $neg(r) = \{a \mid (not\ a) \in body(r)\}$ —i.e., the atoms that appear negated in the body of the rule.

Given a rule r , we denote with $ground(r)$ the set of all rules obtained by consistently replacing the variables in r with constants from \mathcal{F} (i.e., the *ground instances* of r).

We identify special types of rules:

- A rule r is *definite* if $neg(r) = \emptyset$;
- A rule r is a *fact* if $neg(r) \cup pos(r) = \emptyset$; for the sake of readability, the fact

$$h \text{ :- } .$$

will be simply written as

$$h.$$

A program P is a set of rules. A program with variables is understood as a shorthand for the set of all ground instances of the rules in P ; we will use the notation:

$$\text{ground}(P) = \bigcup_{r \in P} \text{ground}(r)$$

A program is *definite* if it contains only definite rules.

The answer set semantics of a program (Subsection 2.2) is highly dependent on the truth value of atoms occurring in the negative literals of the program. For later use, we denote with $NANT(P)$ the atoms which appear in NAF literals in P —i.e.,

$$NANT(P) = \{a \mid a \text{ is a ground atom, } \exists r \in \text{ground}(P) : a \in \text{neg}(r)\}.$$

We will also use \mathcal{A}_P to denote the Herbrand base of a program P . For brevity, we will often write \mathcal{A} instead of \mathcal{A}_P .

Example 3

Let us consider the program P_1 containing the rules:

$$\begin{array}{ll} (r_1) \quad \mathbf{q} & :- \quad \mathbf{a}, \text{not } \mathbf{p}. & (r_2) \quad \mathbf{p} & :- \quad \mathbf{a}, \text{not } \mathbf{q}. \\ (r_3) \quad \mathbf{a} & :- \quad \mathbf{b}. & (r_4) \quad \mathbf{b}. & \end{array}$$

The rule r_3 is definite, while the rule r_4 is a fact. For the rule r_1 we have:

- $\text{head}(r_1) = \mathbf{q}$
- $\text{body}(r_1) = \{\mathbf{a}, \text{not } \mathbf{p}\}$
- $\text{pos}(r_1) = \{\mathbf{a}\}$
- $\text{neg}(r_1) = \{\mathbf{p}\}$

For P_1 , we have $NANT(P_1) = \{\mathbf{p}, \mathbf{q}\}$. □

2.2 Answer Set Semantics and Well-Founded Semantics

We will now review two important semantics of logic programs, the answer set semantics and the well-founded semantics. The former is foundational to ASP and the latter is important for the development of our notion of a justification. We will also briefly discuss the basic components of ASP systems.

2.2.1 Interpretations and Models

A (*three-valued*) interpretation I is a pair $\langle I^+, I^- \rangle$, where $I^+ \cup I^- \subseteq \mathcal{A}$ and $I^+ \cap I^- = \emptyset$. Intuitively, I^+ collects the knowledge of the atoms that are known to be true, while I^- collects the knowledge of the atoms that are known to be false. I is a *complete interpretation* if $I^+ \cup I^- = \mathcal{A}$. If I is not complete, then it means that there are atoms whose truth value is *undefined* with respect to I . For convenience, we will often say that an atom a is undefined in I and mean that the truth value of a is undefined in I .

Let P be a program and I be an interpretation. A positive literal a is satisfied by I , denoted by $I \models a$, if $a \in I^+$. A NAF literal *not* a is satisfied by I —denoted by $I \models \text{not } a$ —if $a \in I^-$. A set of literals S is satisfied by I ($I \models S$) if I satisfies each

literal in S . The notion of satisfaction is easily extended to rules and programs as follows. A rule r is satisfied by I if $I \not\models \text{body}(r)$ or $I \models \text{head}(r)$. I is a *model* of a program if it satisfies all its rules. An atom a is *supported* by I in P if there exists $r \in P$ such that $\text{head}(r) = a$ and $I \models \text{body}(r)$.

We introduce two partial orders on the set of interpretations:

- For two interpretations I and J , we say that $I \sqsubseteq J$ iff $I^+ \subseteq J^+$ and $I^- \subseteq J^-$
- For two interpretations I and J , we say that $I \preceq J$ iff $I^+ \subseteq J^+$

We will denote with \mathcal{I} the set of all possible interpretations and with \mathcal{C} the set of complete interpretations. An important property (Lloyd 1987) of definite programs is that for each program P there exists a unique model M_P which is \preceq -minimal over \mathcal{C} . M_P is called the *least Herbrand model* of P .

2.2.2 Answer Set Semantics

For an interpretation I and a program P , the *reduct* of P w.r.t. I (denoted by P^I) is the program obtained from P by deleting (i) each rule r such that $\text{neg}(r) \cap I^+ \neq \emptyset$, and (ii) all NAF literals in the bodies of the remaining rules. Formally,

$$P^I = \{\text{head}(r) :- \text{pos}(r) \mid r \in P, \text{neg}(r) \cap I^+ = \emptyset\}$$

Given a complete interpretation I , observe that the program P^I is a definite program. A complete interpretation I is an *answer set* (Gelfond and Lifschitz 1988) of P if I^+ is the least Herbrand model of P^I (Apt and Bol 1994).

Example 4

briefly Let us reconsider the program P_1 in Example 3. If we consider the interpretation $I = \langle \{\mathbf{b}, \mathbf{a}, \mathbf{q}\}, \{\mathbf{p}\} \rangle$, then the reduct P_1^I will contain the rules:

$$\begin{array}{l} \mathbf{q} \quad :- \quad \mathbf{a}. \qquad \mathbf{a} \quad :- \quad \mathbf{b}. \\ \mathbf{b}. \end{array}$$

It is easy to see that $\{\mathbf{a}, \mathbf{b}, \mathbf{q}\}$ is the least Herbrand model of this program; thus, I is an answer set of P_1 . \square

For a definite program P and an interpretation I , the immediate consequence operator T_P is defined by

$$T_P(I) = \{a \mid \exists r \in P, \text{head}(r) = a, I \models \text{body}(r)\}.$$

T_P is monotone and has a least fixpoint (van Emden and Kowalski 1976). The fixpoint of T_P will be denoted by $\text{lfp}(T_P)$.

2.2.3 Well-Founded Semantics

Let us describe the *well-founded semantics*, following the definition proposed in (Apt and Bol 1994). We note that this definition is slightly different from the original definition of the well-founded semantics in (Van Gelder et al. 1991). Let us start by recalling some auxiliary definitions.

Definition 1

Let P be a program, S and V be sets of atoms from \mathcal{A} . The set $T_{P,V}(S)$ (*immediate consequence of S w.r.t P and V*) is defined as follows:

$$T_{P,V}(S) = \{a \mid \exists r \in P, \text{head}(r) = a, \text{pos}(r) \subseteq S, \text{neg}(r) \cap V = \emptyset\}$$

It is easy to see that, if V is fixed, the operator is monotone with respect to S . Against, we use $lfp(\cdot)$ to denote the least fixpoint of this operator when V is fixed.

Definition 2

Let P be a program and P^+ be the set of definite rules in P . The sequence $(K_i, U_i)_{i \geq 0}$ is defined as follows:

$$\begin{aligned} K_0 &= lfp(T_{P^+}) & U_0 &= lfp(T_{P, K_0}) \\ K_i &= lfp(T_{P, U_{i-1}}) & U_i &= lfp(T_{P, K_i}) \end{aligned}$$

Let j be the first index of the computation such that $\langle K_j, U_j \rangle = \langle K_{j+1}, U_{j+1} \rangle$. We will denote with $WFP = \langle W^+, W^- \rangle$ the (unique) *well-founded* model of P , where $W^+ = K_j$ and $W^- = \mathcal{A} \setminus U_j$.

briefly

Example 5

Let us reconsider the program P_1 of Example 3. The computation of the well-founded model proceeds as follows:

$$\begin{aligned} K_0 &= \{\mathbf{b}, \mathbf{a}\} \\ U_0 &= \{\mathbf{a}, \mathbf{b}, \mathbf{p}, \mathbf{q}\} \\ K_1 &= \{\mathbf{a}, \mathbf{b}\} = K_0 \\ U_1 &= \{\mathbf{a}, \mathbf{b}, \mathbf{p}, \mathbf{q}\} = U_0 \end{aligned}$$

Thus, the well-founded model will be $\langle \{\mathbf{a}, \mathbf{b}\}, \emptyset \rangle$. Observe that both \mathbf{p} and \mathbf{q} are undefined in the well-founded model. \square

2.3 Answer Set Programming Systems

Several efficient ASP solvers have been developed, such as SMOBELS (Niemelä and Simons 1997), DLV (Eiter et al. 1998), CMOBELS (Giunchiglia et al. 2004), ASSAT (Lin and Zhao 2002), and CLASP (Gebser et al. 2007). One of the most popular ASP solvers is SMOBELS (Niemelä and Simons 1997; Simons et al. 2002) which comes with LPARSE, a grounder. LPARSE takes as input a logic program P and produces as output a simplified version of $ground(P)$. The output of LPARSE is in turn accepted by SMOBELS, and used to produce the answer sets of P (see Figure 1).

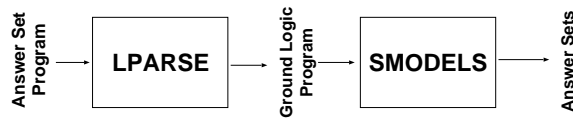


Fig. 1. The LPARSE/SMOBELES System

The LPARSE/SMODELS system supports several extended types of literals, such as the *cardinality literals*, which are of the form: $L \{l_1, \dots, l_n\} U$, where L and U are integers, $L \leq U$, and l_1, \dots, l_n are literals. The cardinality literal is satisfied by an answer set M if the number x of literals in $\{l_1, \dots, l_n\}$ that are true in M is such that $L \leq x \leq U$.

The back-end engine, SMODELS in Figure 1, produces the collection of answer sets of the input program. Various control options can be provided to guide the computation—e.g., establish a limit on the number of answer sets provided or request the answer set to contain specific atoms.

We note that all of the available ASP solvers (Anger et al. 2005; Eiter et al. 1998; Gebser et al. 2007; Giunchiglia et al. 2004; Lin and Zhao 2002) operate in a similar fashion as SMODELS. DLV uses its own grounder while others use LPARSE. New grounder programs have also been recently proposed, e.g., Gringo in (Gebser et al. 2007). SAT-based answer set solvers rely on SAT-solver in computing answer sets (Giunchiglia et al. 2004; Lin and Zhao 2002).

3 Explanations

The traditional methodology employed in ASP relies on encoding each problem Q as a logic program $\pi(Q)$, whose answer sets are in one-to-one correspondence with the solutions of Q . From the software development perspective, it would be important to address the question “*why is M an answer set of the program P ?*” This question gives rise to the question “*why does an atom a belong to M^+ (or M^-)?*” Answering this question can be very important, in that it provides us with explanations regarding the presence (or absence) of different atoms in M . Intuitively, we view answering these questions as the “declarative” parallel of answering questions of the type “*why is 3.1415 the value of the variable x ?*” in the context of imperative languages—a question that can be typically answered by producing and analyzing an *execution trace* (or *event trace* (Auguston 2000)).

The objective of this section is to develop the notion of *explanation*, as a graph structure used to describe the “reason” for the truth value of an atom w.r.t. a given answer set. In particular, each explanation graph will describe the derivation of the truth value (i.e., true or false) of an atom using the rules in the program. The explanation will also need to be flexible enough to explain those contradictory situations, arising during the construction of answer sets, where an atom is made true *and* false at the same time—for reference, these are the situations that trigger a backtracking in systems like SMODELS (Simons et al. 2002).

In the rest of this section, we will introduce this graph-based representation of the support for the truth values of atoms in an interpretation. In particular, we will incrementally develop this representation. We will start with a generic graph structure (*Explanation Graph*), which describes truth values without accounting for program rules. We will then identify specific graph patterns that can be derived from program rules (*Local Consistent Explanations*), and impose them on the explanation graph, to obtain the (J, A) -based *Explanation Graphs*. These graphs are used to explain the truth values of an atom w.r.t. an interpretation J and a set of

assumptions A —where an assumption is an atom for which we will not seek any explanations. The assumptions derive from the inherent “guessing” process involved in the definition of answer sets (and in their algorithmic construction), and they will be used to justify atoms that have been “guessed” in the construction of the answer set and for which a meaningful explanation cannot be constructed.

Before we proceed, let us introduce notation that will be used in the following discussion.

For an atom a , we write a^+ to denote the fact that the atom a is true, and a^- to denote the fact that a is false. We will call a^+ and a^- the *annotated* versions of a . Furthermore, we will define $atom(a^+) = a$ and $atom(a^-) = a$. For a set of atoms S , we define the following sets of annotated atoms:

- $S^p = \{a^+ \mid a \in S\}$,
- $S^n = \{a^- \mid a \in S\}$.

Furthermore, we denote with $not\ S$ the set $not\ S = \{not\ a \mid a \in S\}$.

3.1 Explanation Graphs

In building the notion of justification, we will start from a very general (labeled, directed) graph structure, called *explanation graph*. We will incrementally construct the notion of justification, by progressively adding the necessary restrictions to it.

Definition 3 (Explanation Graph)

For a program P , an *explanation graph* (or *e-graph*) is a labeled, directed graph (N, E) , where $N \subseteq \mathcal{A}^p \cup \mathcal{A}^n \cup \{assume, \top, \perp\}$ and $E \subseteq N \times N \times \{+, -\}$, which satisfies the following properties:

1. the only sinks in the graph are: *assume*, \top , and \perp ;
2. for every $b \in N \cap \mathcal{A}^p$, we have that $(b, assume, -) \notin E$ and $(b, \perp, -) \notin E$;
3. for every $b \in N \cap \mathcal{A}^n$, we have that $(b, assume, +) \notin E$ and $(b, \top, +) \notin E$;
4. for every $b \in N$, if $(b, l, s) \in E$ for some $l \in \{assume, \top, \perp\}$ and $s \in \{+, -\}$ then (b, l, s) is the only outgoing edge originating from b .

Property (1) indicates that each atom appearing in an e-graph should have outgoing edges (which will explain the truth value of the atom). Properties (2) and (3) ensure that true (false) atoms are not explained using explanations that are proper for false (true) atoms. Finally, property (4) ensures that atoms explained using the special explanations *assume*, \top , \perp have only one explanation in the graph. Intuitively,

- \top will be employed to explain program facts—i.e., their truth does not depend on other atoms;
- \perp will be used to explain atoms that do not have defining rules—i.e., the falsity is not dependent on other atoms; and
- *assume* is used for atoms we are not seeking any explanations for.

Each edge of the graph connects two annotated atoms or an annotated atom with one of the nodes in $\{\top, \perp, assume\}$, and it is marked by a label from $\{+, -\}$. Edges

labeled '+' are called *positive* edges, while those labeled '-' are called *negative* edges. A path in an e-graph is *positive* if it contains only positive edges, while a path is *negative* if it contains at least one negative edge. We will denote with $(n_1, n_2) \in E^{*,+}$ the fact that there is a positive path in the e-graph from n_1 to n_2 .

Example 6

Figure 2 illustrates several simple e-graphs. Intuitively,

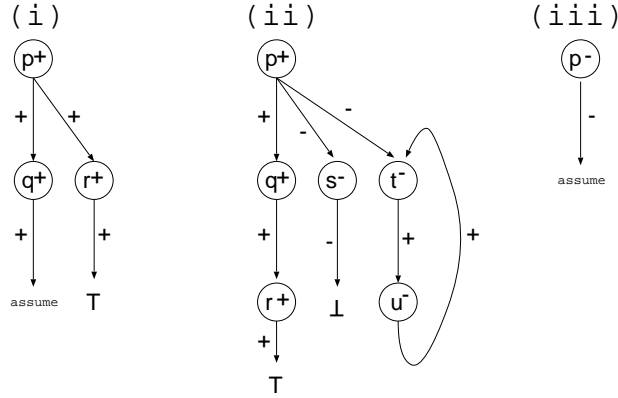


Fig. 2. Simple e-graphs

- The graph (i) describes the true state of p by making it positively dependent on the true state of q and r ; in turn, q is simply assumed to be true while r is a fact in the program.
- The graph (ii) describes more complex dependencies; in particular, observe that t and u are both false and they are mutually dependent—as in the case of a program containing the rules

$$t \text{ :- } u. \quad u \text{ :- } t.$$

Observe also that s is explained being false because there are no rules defining it.

- The graph (iii) states that p has been simply assumed to be false.

□

Given an explanation graph and an atom, we can extract from the graph the elements that directly contribute to the truth value of the atom. We will call this set of elements the *support* of the atom. This is formally defined as follows.

Definition 4

Let $G = (N, E)$ be an e-graph and $b \in N \cap (\mathcal{A}^p \cup \mathcal{A}^n)$ a node in G . The direct support of b in G , denoted by $support(b, G)$, is defined as follows.

- $support(b, G) = \{atom(c) \mid (b, c, +) \in E\} \cup \{not\ atom(c) \mid (b, c, -) \in E\}$, if for every $\ell \in \{assume, \top, \perp\}$ and $s \in \{+, -\}$, $(b, \ell, s) \notin E$;
- $support(b, G) = \{\ell\}$ if $(b, \ell, s) \in E$, $\ell \in \{assume, \top, \perp\}$ and $s \in \{+, -\}$.

Example 7

If we consider the e-graph (ii) in Figure 2, then we have that $support(\mathbf{p}^+, G_2) = \{\mathbf{q}, \text{not } \mathbf{s}, \text{not } \mathbf{t}\}$ while $support(\mathbf{t}^-, G_2) = \{\mathbf{u}\}$.

We also have $support(\mathbf{p}^+, G_1) = \{\mathbf{q}, \mathbf{r}\}$. □

It is worth mentioning that an explanation graph is a general concept aimed at providing arguments for answering the question ‘*why is an atom true or false?*’ In this sense, it is similar to the concept of a support graph used in program analysis (Saha and Ramakrishnan 2005). The main difference between these two concepts lies in that support graphs are defined only for definite programs while explanation graphs are defined for general logic programs. Furthermore, a support graph contains information about the support for *all* answer while an explanation graph stores only the support for *one* atom. An explanation graph can be used to answer the question of why an atom is false which is not the case for support graphs.

3.2 Local Explanations and (J, A) -based Explanations

The next step towards the definition of the concept of justification requires enriching the general concept of e-graph with explanations of truth values of atoms that are derived from the rules of the program.

A *Local Consistent Explanation (LCE)* describes one step of justification for a literal. Note that our notion of local consistent explanation is similar in spirit, but different in practice, from the analogous definition used in (Pemmasani et al. 2004; Roychoudhury et al. 2000). It describes the possible local reasons for the truth/falsity of a literal. If a is true, the explanation contains those bodies of the rules for a that are satisfied by I . If a is false, the explanation contains sets of literals that are false in I and they falsify all rules for a .

The construction of a LCE is performed w.r.t. a possible interpretation and a set of atoms U —the latter contains atoms that are automatically assumed to be false, without the need of justifying them. The need for this last component (to be further elaborated later in the paper) derives from the practice of computing answer sets, where the truth value of certain atoms is first guessed and then later verified.

Definition 5 (Local Consistent Explanation)

Let P be a program, b be an atom, J a possible interpretation, U a set of atoms (*assumptions*), and $S \subseteq \mathcal{A} \cup \text{not } \mathcal{A} \cup \{\text{assume}, \top, \perp\}$ a set of literals. We say that

1. S is a local consistent explanation of b^+ w.r.t. (J, U) , if $b \in J^+$ and
 - $S = \{\text{assume}\}$, or
 - $S \cap \mathcal{A} \subseteq J^+$, $\{c \mid \text{not } c \in S\} \subseteq J^- \cup U$, and there is a rule r in P such that $head(r) = b$ and $S = body(r)$; for convenience, we write $S = \{\top\}$ to denote the case where $body(r) = \emptyset$.
2. S is a local consistent explanation of b^- w.r.t. (J, U) if $b \in J^- \cup U$ and

- $S = \{assume\}$; or
- $S \cap \mathcal{A} \subseteq J^- \cup U$, $\{c \mid not\ c \in S\} \subseteq J^+$, and S is a minimal set of literals such that for every rule $r \in P$, if $head(r) = b$, then $pos(r) \cap S \neq \emptyset$ or $neg(r) \cap \{c \mid not\ c \in S\} \neq \emptyset$; for convenience, we write $S = \{\perp\}$ to denote the case $S = \emptyset$.

We will denote with $LCE_P^p(b, J, U)$ the set of all the LCEs of b^+ w.r.t. (J, U) , and with $LCE_P^n(b, J, U)$ the set of all the LCEs of b^- w.r.t. (J, U) .

Observe that U is the set of atoms that are assumed to be false. For this reason, negative LCEs are defined for elements $J^- \cup U$ but positive LCEs are defined only for elements in J^+ . We illustrate this definition in a series of examples.

Example 8

Let P be the program:

$$\begin{array}{ll} \mathbf{p} & :- \quad \mathbf{q}, \mathbf{r}. & \mathbf{q}. \\ \mathbf{q} & :- \quad \mathbf{r}. & \mathbf{r}. \end{array}$$

The program admits only one answer set $M = \langle \{\mathbf{p}, \mathbf{q}, \mathbf{r}\}, \emptyset \rangle$. The LCEs for the atoms of this program w.r.t. (M, \emptyset) are:

$$\begin{aligned} LCE_P^p(\mathbf{p}, M, \emptyset) &= \{\{\mathbf{q}, \mathbf{r}\}, \{assume\}\} \\ LCE_P^p(\mathbf{q}, M, \emptyset) &= \{\{\top\}, \{\mathbf{r}\}, \{assume\}\} \\ LCE_P^p(\mathbf{r}, M, \emptyset) &= \{\{\top\}, \{assume\}\} \end{aligned}$$

□

The above example shows a program with a unique answer set. The next example discusses the definition in a program with more than one answer set and an empty well-founded model. It also highlights the difference between the positive and negative LCEs for atoms given a partial interpretation and a set of assumptions.

Example 9

Let P be the program:

$$\mathbf{p} \quad :- \quad not\ \mathbf{q}. \qquad \mathbf{q} \quad :- \quad not\ \mathbf{p}.$$

Let us consider the partial interpretation $M = \langle \{\mathbf{p}\}, \emptyset \rangle$. The following are LCEs w.r.t. (M, \emptyset) :

$$\begin{aligned} LCE_P^p(\mathbf{p}, M, \emptyset) &= \{\{assume\}\} \\ LCE_P^n(\mathbf{q}, M, \emptyset) &= LCE_P^p(\mathbf{q}, M, \emptyset) = \{\{\perp\}\} \end{aligned}$$

The above LCEs are explanations for the truth value of \mathbf{p} and \mathbf{q} being true and false with respect to M and the empty set of assumptions. Thus, the only explanation for \mathbf{p} being true is that it is assumed to be true, since the only way to derive \mathbf{p} to be true is to use the first rule and nothing is assumed to be false, i.e., $not\ \mathbf{q}$ is not true. On the other hand, $\mathbf{q} \notin M^- \cup \emptyset$ leads to the fact that there is no explanation for \mathbf{q} being false. Likewise, because $\mathbf{q} \notin M^+$, there is no positive LCE for \mathbf{q} w.r.t. (M, \emptyset) .

The LCEs w.r.t. $(M, \{\mathbf{q}\})$ are:

$$\begin{aligned} LCE_P^p(\mathbf{p}, M, \{\mathbf{q}\}) &= \{\{assume\}, \{not \mathbf{q}\}\} \\ LCE_P^p(\mathbf{q}, M, \{\mathbf{q}\}) &= \{\{assume\}, \{not \mathbf{p}\}\} \end{aligned}$$

Assuming that \mathbf{q} is false leads to one additional explanation for \mathbf{p} being true. Furthermore, there are now two explanations for \mathbf{q} being false. The first one is that it is assumed to be false and the second one satisfies the second condition in Definition 5.

Consider the complete interpretation $M' = \langle \{\mathbf{p}\}, \{\mathbf{q}\} \rangle$. The LCEs w.r.t. (M', \emptyset) are:

$$\begin{aligned} LCE_P^p(\mathbf{p}, M', \emptyset) &= \{\{assume\}, \{not \mathbf{q}\}\} \\ LCE_P^p(\mathbf{q}, M', \emptyset) &= \{\{assume\}, \{not \mathbf{p}\}\} \end{aligned}$$

□

The next example uses a program with a non-empty well-founded model.

Example 10

Let P be the program:

$$\begin{array}{lll} \mathbf{a} & :- & \mathbf{f}, not \mathbf{b}. & \mathbf{b} & :- & \mathbf{e}, not \mathbf{a}. & \mathbf{e}. \\ \mathbf{f} & :- & \mathbf{e}. & \mathbf{d} & :- & \mathbf{c}, \mathbf{e}. & \mathbf{c} & :- & \mathbf{d}, \mathbf{f}. \end{array}$$

This program has the answer sets:

$$M_1 = \langle \{\mathbf{f}, \mathbf{e}, \mathbf{b}\}, \{\mathbf{a}, \mathbf{c}, \mathbf{d}\} \rangle \quad M_2 = \langle \{\mathbf{f}, \mathbf{e}, \mathbf{a}\}, \{\mathbf{c}, \mathbf{b}, \mathbf{d}\} \rangle$$

Observe that the well-founded model of this program is $\langle W^+, W^- \rangle = \langle \{\mathbf{e}, \mathbf{f}\}, \{\mathbf{c}, \mathbf{d}\} \rangle$. The following are LCEs w.r.t. the answer set M_1 and the empty set of assumptions (those for (M_2, \emptyset) have a similar structure):

$$\begin{aligned} LCE_P^p(\mathbf{a}, M_1, \emptyset) &= \{\{not \mathbf{b}\}, \{assume\}\} \\ LCE_P^p(\mathbf{b}, M_1, \emptyset) &= \{\{\mathbf{e}, not \mathbf{a}\}, \{assume\}\} \\ LCE_P^p(\mathbf{e}, M_1, \emptyset) &= \{\{\top\}, \{assume\}\} \\ LCE_P^p(\mathbf{f}, M_1, \emptyset) &= \{\{\mathbf{e}\}, \{assume\}\} \\ LCE_P^p(\mathbf{d}, M_1, \emptyset) &= \{\{\mathbf{c}\}, \{assume\}\} \\ LCE_P^p(\mathbf{c}, M_1, \emptyset) &= \{\{\mathbf{d}\}, \{assume\}\} \end{aligned}$$

□

Let us open a brief parenthesis to discuss some complexity issues related to the existence of LCEs. First, checking whether or not there is a LCE of b^+ w.r.t. (J, U) is equivalent to checking whether or not the program contains a rule r whose head is b and whose body is satisfied by the interpretation $\langle J^+, J^- \cup U \rangle$. This leads to the following observation.

Observation 1

Given a program P , a possible interpretation J , a set of assumptions U , and an atom b , determining whether or not there is a LCE S of b^+ w.r.t. (J, U) such that $S \neq \{assume\}$ can be done in time polynomial in the size of P .

In order to determine whether or not there exists a LCE of b^- w.r.t. (J, U) , we need to find a minimal set of literals S that satisfies the second condition of Definition

5. This can also be accomplished in time polynomial in the size of P . In fact, let P_b be the set of rules in P whose head is b . Furthermore, for a rule r , let

$$S_r(J, U) = \{a \mid a \in \text{pos}(r) \cap (J^- \cup U)\} \cup \{\text{not } a \mid a \in J^+ \cap \text{neg}(r)\}.$$

Intuitively, $S_r(J, U)$ is the maximal set of literals that falsifies the rule r w.r.t. (J, U) . To find a LCE for b^- , it is necessary to have $S_r(J, U) \neq \emptyset$ for every $r \in P_b$. Clearly, computing $S_r(J, U)$ for $r \in P_b$ can be done in polynomial time in the size of P . Finding a minimal set S such that $S \cap S_r \neq \emptyset$ for every $r \in P_b$ can be done by scanning through the set P_b and adding to S (initially set to \emptyset) an arbitrary element of $S_r(J, U)$ if $S \cap S_r(J, U) = \emptyset$. This leads to the following observation.

Observation 2

Given a program P , a possible interpretation J , a set of assumptions U , and an atom b , determining whether there exists a LCE S of b^- w.r.t. (J, U) such that $S \neq \{\text{assume}\}$ can be done in time polynomial in the size of P .

We are now ready to instantiate the notion of e-graph by forcing the edges of the e-graph to represent encodings of local consistent explanations of the corresponding atoms. To select an e-graph as an acceptable explanation, we need two additional components: the current interpretation (J) and the collection (U) of elements that have been introduced in the interpretation without any “supporting evidence”. An e-graph based on (J, U) is defined next.

Definition 6 ((J, U)-Based Explanation Graph)

Let P be a program, J a possible interpretation, U a set of atoms, and b an element in $\mathcal{A}^p \cup \mathcal{A}^n$. A (J, U) -based explanation graph $G = (N, E)$ of b is an e-graph such that

- **(Relevance)** every node $c \in N$ is reachable from b
- **(Correctness)** for every $c \in N \setminus \{\text{assume}, \top, \perp\}$, $\text{support}(c, G)$ is an LCE of c w.r.t. (J, U)

The two additional conditions we impose on the e-graph force the graph to be connected w.r.t. the element b we are justifying, and force the selected nodes and edges to reflect local consistent explanations for the various elements.

The next condition we impose on the explanation graph is aimed at ensuring that no positive cycles are present. The intuition is that atoms that are true in an answer set should have a non-cyclic support for their truth values. Observe that the same does not happen for elements that are false—as in the case of elements belonging to unfounded sets (Apt and Bol 1994).

Definition 7 (Safety)

A (J, U) -based e-graph (N, E) is *safe* if $\forall b^+ \in N, (b^+, b^+) \notin E^{*,+}$.

Example 11

Consider the e-graphs in Figure 3, for the program of Example 10.

Neither the e-graph of a^+ ((i) nor the e-graph (ii)) is a $(M_1, \{c, d\})$ -based e-graph of a^+ , since $\text{support}(b, G) = \{\text{assume}\}$ in both cases, and this does not represent a valid LCE for b^- (since $b \notin M_1^- \cup \{c, d\}$). Observe, on the other hand, that they are both acceptable $(M_2, \{b, c, d\})$ -based e-graphs of a^+ .

The e-graph of c^+ (the graph (iii)) is neither a $(M_1, \{c, d\})$ -based nor a $(M_2, \{b, c, d\})$ -based e-graph of c^+ , while the e-graph of c^- (graph (iv)) is a $(M_1, \{c, d\})$ -based and a $(M_2, \{b, c, d\})$ -based e-graph of c^- .

Observe also that all the graphs are safe. \square

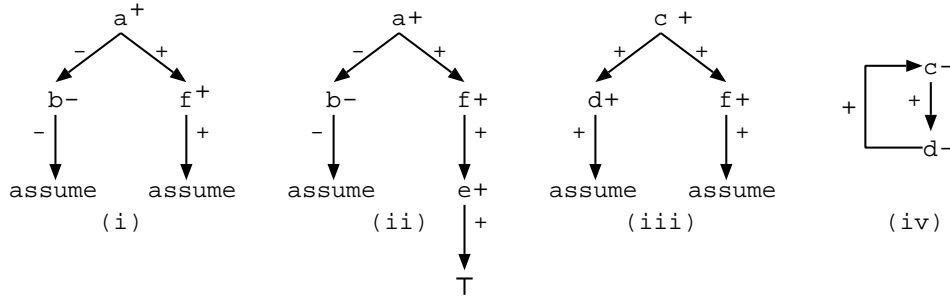


Fig. 3. Sample (J, U) -based Explanation Graphs

4 Off-Line Justifications

Off-line justifications are employed to characterize the “reason” for the truth value of an atom w.r.t. a given answer set M . The definition will represent a refinement of the (M, A) -based explanation graph, where A will be selected according to the properties of the answer set M . Off-line justifications will rely on the assumption that M is a *complete* interpretation.

Let us start with a simple observation. If M is an answer set of a program P , and WF_P is the well-founded model of P , then it is known that, $WF_P^+ \subseteq M^+$ and $WF_P^- \subseteq M^-$ (Apt and Bol 1994). Furthermore, we observe that the content of M is uniquely determined by the truth values assigned to the atoms in $V = NANT(P) \setminus (WF_P^+ \cup WF_P^-)$, i.e., the atoms that

- appear in negative literals in the program, and
- their truth value is not determined by the well-founded model.

We are interested in the subsets of V with the following property: if all the elements in the subset are assumed to be false, then the truth value of all other atoms in A is uniquely determined and leads to the desired answer set. We call these subsets the *assumptions* of the answer set. Let us characterize this concept more formally.

Definition 8 (Tentative Assumptions)

Let P be a program and M be an answer set of P . The *tentative assumptions* of P w.r.t. M (denoted by $\mathcal{TA}_P(M)$) are defined as:

$$\mathcal{TA}_P(M) = \{a \mid a \in \text{NANT}(P) \wedge a \in M^- \wedge a \notin (WF_P^+ \cup WF_{\bar{P}}^-)\}$$

The negative reduct of a program P w.r.t. a set of atoms U is a program obtained from P by forcing all the atoms in U to be false.

Definition 9 (Negative Reduct)

Let P be a program, M an answer set of P , and $U \subseteq \mathcal{TA}_P(M)$ a set of tentative assumption atoms. The *negative reduct* of P w.r.t. U , denoted by $NR(P, U)$, is the set of rules:

$$NR(P, U) = P \setminus \{r \mid \text{head}(r) \in U\}.$$

Example 12

Let us consider the program

```

p :- not q.           q :- not p.
r :- p, s.           t :- q, u.
s.

```

The well-founded model for this program is $\langle \{\mathbf{s}\}, \{\mathbf{u}\} \rangle$. The program has two answer sets, $M_1 = \langle \{\mathbf{p}, \mathbf{s}, \mathbf{r}\}, \{\mathbf{t}, \mathbf{u}, \mathbf{q}\} \rangle$ and $M_2 = \langle \{\mathbf{q}, \mathbf{s}\}, \{\mathbf{p}, \mathbf{r}, \mathbf{t}, \mathbf{u}\} \rangle$. The tentative assumptions for this program w.r.t. M_1 is the set $\{\mathbf{q}\}$. If we consider the set $\{\mathbf{q}\}$, the negative reduct of the program is the set of rules

```

p :- not q.
r :- p, s.       t :- q, u.
s.

```

□

We are now ready to introduce the proper concept of assumptions—these are those tentative assumptions that are sufficient to allow the reconstruction of the answer set.

Definition 10 (Assumptions)

Let P be a program and M be an answer set of P . An *assumption* w.r.t. M is a set of atoms U satisfying the following properties:

- (1) $U \subseteq \mathcal{TA}_P(M)$, and
- (2) the well-founded model of $NR(P, U)$ is equal to M —i.e.,

$$WF_{NR(P, U)} = M.$$

We will denote with $\text{Assumptions}(P, M)$ the set of all assumptions of P w.r.t. M . A *minimal assumption* is an assumption that is minimal w.r.t. the set inclusion operator. We will denote with $\mu\text{Assumptions}(P, M)$ the set of all the minimal assumptions of P w.r.t. M .

An important observation we can make is that $\text{Assumptions}(P, M)$ is not an empty set, since the complete set $\mathcal{TA}_P(M)$ is an assumption.

Proposition 1

Given a program P and an answer set M of P , the well-founded model of the program $NR(P, \mathcal{TA}_P(M))$ is equal to M .

Proof

Appendix A. \square

Example 13

Let us consider the program of Example 9. The interpretation $M = \langle \{p\}, \{q\} \rangle$ is an answer set. For this program we have:

$$\begin{aligned} WF_P &= \langle \emptyset, \emptyset \rangle \\ \mathcal{TA}_P(\langle \{p\}, \{q\} \rangle) &= \{q\} \end{aligned}$$

Observe that $NR(P, \{q\}) = \{p :- \text{not } q\}$. The well-founded model of this program is $\langle \{p\}, \{q\} \rangle$, which is equal to M . Thus, $\{q\}$ is an assumption of P w.r.t. M . In particular, one can see that this is the only assumption we can have. \square

Example 14

Let us consider the following program P :

$$\begin{array}{lll} a & :- & f, \text{ not } b. & b & :- & e, \text{ not } a. & e. \\ f & :- & e. & d & :- & c, e. & c & :- & d, f, \text{ not } k. \\ k & :- & a. & & & & & & \end{array}$$

The interpretation $M_1 = \langle \{f, e, b\}, \{a, c, d, k\} \rangle$ is an answer set of the program. In particular:

$$\begin{aligned} WF_P &= \langle \{e, f\}, \{d, c\} \rangle \\ \mathcal{TA}_P(\langle \{f, e, b\}, \{a, c, d\} \rangle) &= \{a, k\} \end{aligned}$$

The program $NR(P, \{a\})$ is:

$$\begin{array}{lll} b & :- & e, \text{ not } a. & e. \\ f & :- & e. & d & :- & c, e. \\ c & :- & d, f, \text{ not } k. & k & :- & a. \end{array}$$

The well-founded model of this program is $\langle \{e, f, b\}, \{a, c, d, k\} \rangle$. Thus, $\{a\}$ is an assumption w.r.t. M_1 .

Observe also that if we consider $NR(P, \{a, k\})$

$$\begin{array}{lll} b & :- & e, \text{ not } a. & e. \\ f & :- & e. & d & :- & c, e. \\ c & :- & d, f, \text{ not } k. & & & \end{array}$$

The well-founded model of this program is also $\langle \{e, f, b\}, \{a, c, d, k\} \rangle$, thus making $\{a, k\}$ another assumption. Note that this second assumption is not minimal. \square

We will now specialize e-graphs to the case of answer sets, where only false elements can be used as assumptions.

Definition 11 (Off-line Explanation Graph)

Let P be a program, J a partial interpretation, U a set of atoms, and b an element in $\mathcal{A}^p \cup \mathcal{A}^n$. An *off-line explanation graph* $G = (N, E)$ of b w.r.t. J and U is a (J, U) -based e-graph of b satisfying the following additional conditions:

- there exists no $p^+ \in N$ such that $(p^+, \text{assume}, +) \in E$; and
- $(p^-, \text{assume}, -) \in E$ iff $p \in U$.

We will denote with $\mathcal{E}(b, J, U)$ the set of all off-line explanation graphs of b w.r.t. J and U .

The first condition ensures that true elements cannot be treated as assumptions, while the second condition ensures that only assumptions are justified as such in the graph.

Definition 12 (Off-line Justification)

Let P be a program, M an answer set, $U \in \text{Assumptions}(P, M)$, and $a \in \mathcal{A}^p \cup \mathcal{A}^n$. An *off-line justification* of a w.r.t. M and U is an element (N, E) of $\mathcal{E}(a, M, U)$ which is safe.

If M is an answer set and $x \in M^+$ (resp. $x \in M^-$), then G is an off-line justification of x w.r.t. M and the assumption U iff G is an off-line justification of x^+ (resp. x^-) w.r.t. M and U .

Example 15

Let us consider the program in Example 10. We have that $NANT(P) = \{b, a\}$. The assumptions for this program are:

$$\text{Assumptions}(P, M_1) = \{\{a\}\} \quad \text{and} \quad \text{Assumptions}(P, M_2) = \{\{b\}\}.$$

The off-line justifications for atoms in M_1 w.r.t. M_1 and $\{a\}$ are shown in Figure 4.

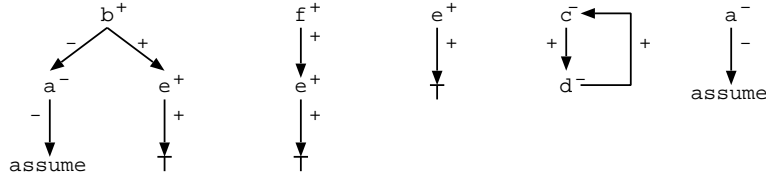


Fig. 4. Off-line justifications w.r.t. M_1 and $\{a\}$ for b^+ , f^+ , e^+ , c^- and a^- (left to right)

Justifications are built by assembling items from the LCEs of the various atoms and avoiding the creation of positive cycles in the justification of true atoms. Also, the justification is built w.r.t. a chosen set of assumptions (A), whose elements are all assumed false.

In general, an atom may admit multiple justifications, even w.r.t. the same assumptions. The following lemma shows that elements in WF_P can be justified without negative cycles and assumptions.

Lemma 4.1

Let P be a program, M an answer set, and WF_P the well-founded model of P . Each atom $a \in WF_P$ has a justification w.r.t. M and \emptyset which does not contain any negative cycle.

From the definition of assumption and from the previous lemma we can infer that a justification free of negative cycles can be built for every atom.

Proposition 2

Let P be a program and M an answer set. For each atom a , there is an off-line justification w.r.t. M and $M^- \setminus WF_P^-$ which does not contain negative cycles.

Proposition 2 underlines an important property—the fact that all true elements can be justified in a non-cyclic fashion. This makes the justification more natural, reflecting the non-cyclic process employed in constructing the minimal answer set (e.g., using the iterations of T_P) and the well-founded model (e.g., using the characterization in (Brass et al. 2001)). This also gracefully extends a similar property satisfied by the justifications under well-founded semantics used in (Roychoudhury et al. 2000). Note that the only cycles possibly present in the justifications are positive cycles associated to (mutually dependent) false elements—this is an unavoidable situation due the semantic characterization in well-founded and answer set semantics (e.g., unfounded sets). A similar design choice has been made in (Pemasani et al. 2004; Roychoudhury et al. 2000).

Example 16

Let us reconsider the following program P from Example 14:

a	$:-$	$f, \text{ not } b.$	b	$:-$	$e, \text{ not } a.$	$e.$
f	$:-$	$e.$	d	$:-$	$c, e.$	c
k	$:-$	$a.$				$:-$
						$d, f, \text{ not } k.$

and the answer set $M = \langle \{f, e, b\}, \{a, c, d, k\} \rangle$ is an answer set of the program. The well-founded model of this program is

$$WF_P = \langle \{e, f\}, \{d, c\} \rangle$$

a and k are assumed to be false. Off-line justifications for b^+, f^+, e^+ and for c^-, d^-, a^- with respect to M and $M^- \setminus WF_P^- = \{a, k\}$, which do not contain negative cycles, are the same as those depicted in Figure 4. k^- has an off-line justification in which it is connected to *assume* by a negative edge, as it is assumed to be false. \square

5 On-Line Justifications for ASP

Off-line justifications provide a “declarative trace” for the truth values of the atoms present in an answer set. The majority of the inference engines for ASP construct answer sets in an incremental fashion, making choices (and possibly undoing them) and declaratively applying the rules in the program. Unexpected results (e.g., failure to produce any answer sets) require a more refined view of computation. One way

to address this problem is to refine the notion of justification to make possible the “declarative tracing” of atoms w.r.t. a partially constructed interpretation. This is similar to debugging of imperative languages, where breakpoints can be set and the state of the execution explored at any point during the computation. In this section, we introduce the concept of *on-line justification*, which is generated *during* the computation of an answer set and allows us to justify atoms w.r.t. an incomplete interpretation—that represents an intermediate step in the construction of the answer set.

5.1 Computation

The concept of on-line justification is applicable to computation models that construct answer sets in an incremental fashion, e.g., SMOBELS and DLV (Simons et al. 2002; Eiter et al. 1998; Gebser et al. 2007; Anger et al. 2005). We can view the computation as a sequence of steps, each associated to a partial interpretation. We will focus, in particular, on computation models where the progress towards the answer set is monotonic.

Definition 13 (General Computation)

Let P be a program. A *general computation* is a sequence M_0, M_1, \dots, M_k , such that

- (i) $M_0 = \langle \emptyset, \emptyset \rangle$,
- (ii) M_0, \dots, M_{k-1} are partial interpretations, and
- (iii) $M_i \sqsubseteq M_{i+1}$ for $i = 0, \dots, k-1$.

A *general complete computation* is a computation M_0, \dots, M_k such that M_k is an answer set of P .

In general, we do not require M_k —the ending point of the computation—to be a partial interpretation, since we wish to model computations that can also “fail”—i.e., $M_k^+ \cap M_k^- \neq \emptyset$. This is, for example, what might happen during a SMOBELS computation—whenever the **Conflict** function succeeds (Simons et al. 2002).

We will refer to a pair of sets of atoms as a *possible interpretation* (or *p-interpretation* for short). Clearly, each partial interpretation is a p-interpretation, but not vice versa. Abusing the notation, we use J^+ and J^- to indicate the first and second component of a p-interpretation J ; moreover, $I \sqsubseteq J$ denotes that $I^+ \subseteq J^+$ and $I^- \subseteq J^-$.

Our objective is to associate a form of justification to each intermediate step M_i of a general computation. Ideally, we would like the justifications associated to each M_i to explain truth values in the “same way” as in the final off-line justification. Since the computation model might rely on guessing some truth values, M_i might not contain sufficient information to develop a valid justification for each element in M_i . We will identify those atoms for which a justification can be constructed given M_i . These atoms describe a p-interpretation $D_i \sqsubseteq M_i$. The computation of D_i is defined based on the two operators, Γ and Δ , which will respectively compute D_i^+ and D_i^- .

Let us start with some preliminary definitions. Let P be a program and I be a p-interpretation. A set of atoms S is called a *cycle w.r.t. I* if, for every $a \in S$ and for each $r \in P$ such that $\text{head}(r) = a$, we have that one of the following holds:

- $\text{pos}(r) \cap I^- \neq \emptyset$ (rule is falsified by I), or
- $\text{neg}(r) \cap I^+ \neq \emptyset$ (rule is falsified by I), or
- $\text{pos}(r) \cap S \neq \emptyset$ (rule is in a cycle with elements of S).

We can observe that, if I is an interpretation, S is a cycle w.r.t. I , and M is an answer set with $I \sqsubseteq M$, then $S \subseteq M^-$ —since the elements of S are either falsified by the interpretation (and, thus, by M) or they are part of an unfounded set.

The set of cycles w.r.t. I is denoted by $\text{cycles}(I)$. Furthermore, for every element $a \in \mathcal{A}^p \cup \mathcal{A}^n$, let $PE(a, I)$ be the set of local consistent explanations of a w.r.t. I and \emptyset —i.e., LCEs that do not require any assumptions and that build on the interpretation I .

We are now ready to define the operators that will compute the D_i subset of the p-interpretation M_i .

Definition 14

Let P be a program and $I \sqsubseteq J$ be two p-interpretations. We define

$$\begin{aligned} \Gamma_I(J) &= I^+ \cup \{\text{head}(r) \in J^+ \mid r \in P, I \models \text{body}(r)\} \\ \Delta_I(J) &= I^- \cup \{a \in J^- \mid PE(a^-, I) \neq \emptyset\} \cup \bigcup \{S \mid S \in \text{cycles}(I), S \subseteq J^-\} \end{aligned}$$

Intuitively, for $I \sqsubseteq J$, $\Gamma_I(J)$ is a set of atoms that are true in J and they will remain true in every answer set extending J , if J is a partial interpretation. The set $\Delta_I(J)$ contains atoms that are false in J and in each answer set that extends J . In particular, if I is the set of “justifiable” atoms—i.e., atoms for which we can construct a justification—and J is the result of the current computation step, then we have that $\langle \Gamma_I(J), \Delta_I(J) \rangle$ is a new interpretation satisfying the following two properties:

- $I \sqsubseteq \langle \Gamma_I(J), \Delta_I(J) \rangle \sqsubseteq J$, and
- it is possible to create a justification for all elements in $\langle \Gamma_I(J), \Delta_I(J) \rangle$.

Observe that it is not necessarily true that $\Gamma_I(J) = J^+$ and $\Delta_I(J) = J^-$. This means that there may be elements in the current step of computation for which it is not possible (yet) to construct a justification. This reflects the practice of guessing literals and propagating these guesses in the computation of answer sets, implemented by several solvers (based on variations of the Davis-Putnam-Logemann-Loveland procedure (Davis et al. 1962)).

We are now ready to specify how the set D_i is computed. Let $WF_P = \langle W^+, W^- \rangle$ be the well-founded model of P and let J be a p-interpretation.²

$$\begin{aligned} \Gamma^0(J) &= \Gamma_{\langle \emptyset, \emptyset \rangle}(J) & \Delta^0(J) &= \mathcal{TA}_P(J) \cup \Delta_{\langle \emptyset, \emptyset \rangle}(J) \\ \Gamma^{i+1}(J) &= \Gamma_{I_i}(J) & \Delta^{i+1}(J) &= \Delta_{I_i}(J) \\ &(\text{where } I_i = \langle \Gamma^i(J), \Delta^i(J) \rangle) \end{aligned}$$

² Remember that $\mathcal{TA}_P(J) = \{a \mid a \in \text{NANT}(P) \wedge a \in J^- \wedge a \notin (WF_P^+ \cup WF_P^-)\}$.

Intuitively,

1. The iteration process starts by collecting the facts of P (Γ^0) and all those elements that are false either because there are no defining rules for them or because they have been chosen to be false in the construction of J . All these elements can be easily provided with justifications.
2. The successive iterations expand the set of known justifiable elements from J using Γ and Δ .

Finally, we repeat the iteration process until a fixpoint is reached:

$$\Gamma(J) = \bigcup_{i=0}^{\infty} \Gamma^i(J) \quad \text{and} \quad \Delta(J) = \bigcup_{i=0}^{\infty} \Delta^i(J)$$

Because $\Gamma^i(J) \subseteq \Gamma^{i+1}(J) \subseteq J^+$ and $\Delta^i(J) \subseteq \Delta^{i+1}(J) \subseteq J^-$ (recall that $I \sqsubseteq J$), we know that both $\Gamma(J)$ and $\Delta(J)$ are well-defined. We can prove the following:

Proposition 3

For a program P , we have that:

- Γ and Δ maintains the consistency of J , i.e., if J is an interpretation, then $\langle \Gamma(J), \Delta(J) \rangle$ is also an interpretation;
- Γ and Δ are monotone w.r.t the argument J , i.e., if $J \sqsubseteq J'$ then $\Gamma(J) \subseteq \Gamma(J')$ and $\Delta(J) \subseteq \Delta(J')$;
- $\Gamma(WF_P) = WF_P^+$ and $\Delta(WF_P) = WF_P^-$; and
- If M is an answer set of P , then $\Gamma(M) = M^+$ and $\Delta(M) = M^-$.

We next introduce the notion of on-line explanation graph.

Definition 15 (On-line Explanation Graph)

Let P be a program, A a set of atoms, J a p-interpretation, and $a \in \mathcal{A}^p \cup \mathcal{A}^n$. An *on-line explanation graph* $G = (N, E)$ of a w.r.t. J and A is a (J, A) -based e-graph of a .

In particular, if J is an answer set of P , then any off-line e-graph of a w.r.t. J and A is also an on-line e-graph of a w.r.t. J and A .

Observe that $\Gamma^0(J)$ contains the set of facts of P that belongs to J^+ , while $\Delta^0(J)$ contains the set of atoms without defining rules and atoms belonging to positive cycles of P . As such, it is easy to see that, for each atom a in $\langle \Gamma^0(J), \Delta^0(J) \rangle$, we can construct an e-graph for a^+ or a^- whose nodes belong to $\Gamma^0(J) \cup \Delta^0(J)$. Moreover:

- if $a \in \Gamma^{i+1}(J) \setminus \Gamma^i(J)$, then an e-graph with nodes (except a^+) belonging to $\Gamma^i(J) \cup \Delta^i(J)$ can be constructed;
- if $a \in \Delta^{i+1}(J) \setminus \Delta^i(J)$, an e-graph with nodes (except a^-) belonging to $\Gamma^{i+1}(J) \cup \Delta^{i+1}(J)$ can be constructed.

This leads to the following lemma.

Lemma 5.1

Let P be a program, J a p-interpretation, and $A = \mathcal{TA}_P(J)$. The following properties hold:

- For each atom $a \in \Gamma(J)$ (resp. $a \in \Delta(J)$), there exists a *safe* off-line e-graph of a^+ (resp. a^-) w.r.t. J and A ;
- for each atom $a \in J^+ \setminus \Gamma(J)$ (resp. $a \in J^- \setminus \Delta(J)$) there exists an on-line e-graph of a^+ (resp. a^-) w.r.t. J and A .

We will now discuss how the above proposition can be utilized in defining a notion called *on-line justification*. To this end, we associate to each partial interpretation J a snapshot $S(J)$:

Definition 16

A *snapshot* of a p-interpretation J is a tuple $S(J) = \langle \text{Off}(J), \text{On}(J), D \rangle$, where

- $D = \langle \Gamma(J), \Delta(J) \rangle$,
- For each a in $\Gamma(J)$,
 $\text{Off}(J)$ contains exactly one safe off-line e-graph of a^+ w.r.t. J and $\mathcal{TA}_P(J)$;
- For each a in $\Delta(J)$,
 $\text{Off}(J)$ contains exactly one safe off-line e-graph of a^- w.r.t. J and $\mathcal{TA}_P(J)$;
- For each $a \in J^+ \setminus \Gamma(J)$,
 $\text{On}(J)$ contains exactly one on-line e-graph of a^+ w.r.t. J and $\mathcal{TA}_P(J)$;
- For each $a \in J^- \setminus \Delta(J)$,
 $\text{On}(J)$ contains exactly one on-line e-graph of a^- w.r.t. J and $\mathcal{TA}_P(J)$.

Definition 17 (On-line Justification)

Given a computation M_0, M_1, \dots, M_k , an *on-line justification* of the computation is a sequence of snapshots $S(M_0), S(M_1), \dots, S(M_k)$.

It is worth to point out that an on-line justification can be obtained in answer set solvers employing the computation model described in Definition 13. This will be demonstrated in the next section where we discuss the computation of on-line justifications in the SMODELS system. We next illustrate the concept of an on-line justification.

Example 17

Let us consider the program P containing

$$\begin{array}{lll} \mathbf{s} & :- & \mathbf{a}, \text{ not } \mathbf{t}. & \mathbf{a} & :- & \mathbf{f}, \text{ not } \mathbf{b}. & \mathbf{b} & :- & \mathbf{e}, \text{ not } \mathbf{a}. \\ \mathbf{e}. & & & \mathbf{f} & :- & \mathbf{e}. & & & \end{array}$$

Two possible general computations of P are

$$\begin{array}{llll} M_0^1 = \langle \{\mathbf{e}, \mathbf{s}\}, \emptyset \rangle & \mapsto & M_1^1 = \langle \{\mathbf{e}, \mathbf{s}, \mathbf{a}\}, \{\mathbf{t}\} \rangle & \mapsto & M_2^1 = \langle \{\mathbf{e}, \mathbf{s}, \mathbf{a}, \mathbf{f}\}, \{\mathbf{t}, \mathbf{b}\} \rangle \\ M_0^2 = \langle \{\mathbf{e}, \mathbf{f}\}, \emptyset \rangle & \mapsto & M_1^2 = \langle \{\mathbf{e}, \mathbf{f}\}, \{\mathbf{t}\} \rangle & \mapsto & M_2^2 = \langle \{\mathbf{e}, \mathbf{f}, \mathbf{b}, \mathbf{a}\}, \{\mathbf{t}, \mathbf{a}, \mathbf{b}, \mathbf{s}\} \rangle \end{array}$$

The first computation is a complete computation leading to an answer set of P while the second one is not.

An on-line justification for the first computation is given next:

$$\begin{aligned} S(M_0^1) &= \langle X_0, Y_0, \langle \{\mathbf{e}\}, \emptyset \rangle \rangle \\ S(M_1^1) &= \langle X_0 \cup X_1, Y_0 \cup Y_1, \langle \{\mathbf{e}\}, \{\mathbf{t}\} \rangle \rangle \\ S(M_2^1) &= \langle X_0 \cup X_1 \cup X_2, \emptyset, M_2^1 \rangle \end{aligned}$$

where (for the sake of simplicity we report only the edges of the graphs):

$$\begin{aligned} X_0 &= \{(\mathbf{e}^+, \top, +)\} \\ Y_0 &= \{(\mathbf{s}^+, \text{assume}, +)\} \\ X_1 &= \{(\mathbf{t}^-, \perp, -)\} \\ Y_1 &= \{(\mathbf{a}^+, \text{assume}, +)\} \\ X_2 &= \{(\mathbf{f}^+, \mathbf{e}^+, +), (\mathbf{s}^+, \mathbf{a}^+, +), (\mathbf{s}^+, \mathbf{t}^-, -), (\mathbf{a}^+, \mathbf{f}^+, +), (\mathbf{a}^+, \mathbf{b}^-, -), (\mathbf{b}^-, \text{assume}, -)\} \end{aligned}$$

An on-line justification for the second computation is:

$$\begin{aligned} S(M_0^2) &= \langle X_0, Y_0, \langle \{\mathbf{e}, \mathbf{f}\}, \emptyset \rangle \rangle \\ S(M_1^2) &= \langle X_0 \cup X_1, Y_0, \langle \{\mathbf{e}, \mathbf{f}\}, \{\mathbf{t}\} \rangle \rangle \\ S(M_2^2) &= \langle X_0 \cup X_1 \cup X_2, Y_0 \cup Y_2, M_2^2 \rangle \end{aligned}$$

where:

$$\begin{aligned} X_0 &= \{(\mathbf{e}^+, \top, +), (\mathbf{f}^+, \mathbf{e}^+, +)\} \\ Y_0 &= \emptyset \\ X_1 &= \{(\mathbf{t}^-, \perp, -)\} \\ Y_1 &= \emptyset \\ X_2 &= \{(\mathbf{a}^+, \mathbf{f}^+, +), (\mathbf{a}^+, \mathbf{b}^-, -), (\mathbf{b}^+, \mathbf{e}^+, +), (\mathbf{b}^+, \mathbf{a}^-, -)\} \\ Y_2 &= \{(\mathbf{a}^-, \text{assume}, -), (\mathbf{b}^-, \text{assume}, -)\} \end{aligned}$$

□

We can relate the on-line justifications and off-line justifications as follows.

Lemma 5.2

Let P be a program, J an interpretation, and M an answer set such that $J \sqsubseteq M$. For every atom a , if (N, E) is a safe off-line e-graph of a w.r.t. J and A where $A = J^- \cap \mathcal{TA}_P(M)$ then it is an off-line justification of a w.r.t. M and $\mathcal{TA}_P(M)$.

This leads to the following proposition.

Proposition 4

Let M_0, \dots, M_k be a general complete computation and $S(M_0), \dots, S(M_k)$ be an on-line justification of the computation. Then, for each atom a in M_k , the e-graph of a in $S(M_k)$ is an off-line justification of a w.r.t. M_k and $\mathcal{TA}_P(M_k)$.

6 SMODELS On-line Justifications

The notion of on-line justification presented in the previous section is very general, to fit the needs of different answer set solver implementations that follow the computation model presented in Subsection 5.1. In this section, we illustrate how the notion of on-line justification has been specialized to (and implemented in) a specific computation model—the one used in SMODELS (Simons et al. 2002). This

allows us to define an incremental version of on-line justification—where the specific steps performed by SMOBELS are used to guide the incremental construction of the justification. The choice of SMOBELS was dictated by availability of its source code and its elegant design.

We begin with an overview of the algorithms employed by SMOBELS. The following description has been adapted from (Giunchiglia and Maratea 2005; Simons et al. 2002). Although more abstract than the concrete implementation, and without various implemented features (e.g., heuristics, lookahead), it is sufficiently faithful to capture the spirit of our approach, and to guide the implementation (see Section 6.3).

6.1 An Overview of SMOBELS' Computation

We propose a description of the SMOBELS algorithms based on a composition of state-transformation operators. In the following, we say that an interpretation I does not satisfy the body of a rule r (i.e., $body(r)$ is false in I) if $(pos(r) \cap I^-) \cup (neg(r) \cap I^+) \neq \emptyset$.

ATLEAST Operator:

The *AtLeast* operator is used to expand a partial interpretation I in such a way that each answer set M of P that “agrees” with I —i.e., the elements in I have the same truth value in M (or $I \sqsubseteq M$)—also agrees with the expanded interpretation. Given a program P and a partial interpretation I , we define the intermediate operators AL_P^1, \dots, AL_P^4 as follows:

- **Case 1.** if $r \in P$, $head(r) \notin I^+$, $pos(P) \subseteq I^+$ and $neg(P) \subseteq I^-$ then

$$AL_P^1(I)^+ = I^+ \cup \{head(r)\} \quad \text{and} \quad AL_P^1(I)^- = I^-$$

- **Case 2.** if $a \notin I^+ \cup I^-$ and $\forall r \in P. (head(r) = a \Rightarrow body(r) \text{ is false in } I)$, then

$$AL_P^2(I)^+ = I^+ \quad \text{and} \quad AL_P^2(I)^- = I^- \cup \{a\}$$

- **Case 3.** if $a \in I^+$ and r is the only rule in P with $head(r) = a$ and whose body is not false in I then

$$AL_P^3(I)^+ = I^+ \cup pos(r) \quad \text{and} \quad AL_P^3(I)^- = I^- \cup neg(r)$$

- **Case 4.** if $a \in I^-$, $head(r) = a$, and

— if $pos(r) \setminus I^+ = \{b\}$ then

$$AL_P^4(I)^+ = I^+ \quad \text{and} \quad AL_P^4(I)^- = I^- \cup \{b\}$$

— if $neg(r) \setminus I^+ = \{b\}$ then

$$AL_P^4(I)^+ = I^+ \cup \{b\} \quad \text{and} \quad AL_P^4(I)^- = I^-$$

Given a program P and an interpretation I , $AL_P(I) = AL_P^i(I)$ if $AL_P^i(I) \neq I$ and $\forall j < i. AL_P^j(I) = I$ ($1 \leq i \leq 4$); otherwise, $AL_P(I) = I$.

ATMOST Operator:

The $AtMost_P$ operator recognizes atoms that are defined exclusively as mutual positive dependencies (i.e., “positive loops”)—and falsifies them. Given a set of atoms S , the operator AM_P is defined as $AM_P(S) = S \cup \{head(r) \mid r \in P \wedge pos(r) \subseteq S\}$.

Given an interpretation I , the $AtMost_P(I)$ operator is defined as

$$AtMost_P(I)^+ = I^+ \quad \text{and} \quad AtMost_P(I)^- = I^- \cup \{p \in \mathcal{A} \mid p \notin \bigcup_{i \geq 0} S_i\}$$

where $S_0 = I^+$ and $S_{i+1} = AM_P(S_i)$.

CHOOSE Operator:

This operator is used to randomly select an atom that is unknown in a given interpretation. Given a partial interpretation I , $choose_P$ returns an atom of \mathcal{A} such that

$$choose_P(I) \notin I^+ \cup I^- \quad \text{and} \quad choose_P(I) \in NANT(P) \setminus (WF_P^+ \cup WF_P^-).$$

SMODELS Computation:

Given an interpretation I , we define the transitions:

$$\begin{aligned} I &\mapsto_{AL^c} I' && [\text{If } I' = AL_P^c(I), c \in \{1, 2, 3, 4\} \\ I &\mapsto_{atmost} I' && [\text{If } I' = AtMost_P(I) \\ I &\mapsto_{choice} I' && [\begin{array}{l} \text{If } I' = \langle I^+ \cup \{choose_P(I)\}, I^- \rangle \text{ or} \\ I' = \langle I^+, I^- \cup \{choose_P(I)\} \rangle \end{array} \end{aligned}$$

If there is an α in $\{AL^1, AL^2, AL^3, AL^4, atmost, choice\}$ such that $I \mapsto_\alpha I'$, then we will simply denote this fact with $I \mapsto I'$.

```

function smodels(P):
  S = ⟨∅, ∅⟩;
  loop
    S = expand(P, S);
    if (S+ ∩ S- ≠ ∅) then
      fail;
    if (S+ ∪ S- = A) then
      success(S);
    pick either % non-deterministic choice
      S+ = S+ ∪ {choose(S)} or
      S- = S- ∪ {choose(S)}
  endloop;

```

Fig. 5. Sketch of *smodels*

```

function expand(P, S):
  loop
    S' = S;
    repeat
      S = AL_P(S);
    until (S = AL_P(S));
    S = AtMost(P, S);
    if (S' = S) then return (S);
  endloop;

```

Fig. 6. Sketch of *expand*

The SMODELS system imposes constraints on the order of application of the

transitions. Intuitively, the SMOBELS computation is depicted in the algorithms of Figs. 5 and 6.

We will need the following notations. A computation $I_0 \mapsto I_1 \mapsto I_2 \mapsto \dots \mapsto I_n$ is said to be *AL-pure* if every transition in the computation is an AL^c transitions and for every $c \in \{1, 2, 3, 4\}$, $AL_P^c(I_n) = I_n$. A choice point of a computation $I_0 \mapsto I_1 \mapsto I_2 \mapsto \dots \mapsto I_n$ is an index $1 \leq j < n$ such that $I_j \mapsto_{choice} I_{j+1}$.

Definition 18 (SMOBELES Computation)

Let P be a program. Let

$$C = I_0 \mapsto I_1 \mapsto I_2 \mapsto \dots \mapsto I_n$$

be a computation and

$$0 \leq \nu_1 < \nu_2 < \dots < \nu_r < n$$

($r \geq 0$) be the sequence of all choice points in C . We say that C is a SMOBELES *computation* if for every $0 \leq j \leq r$, there exists a sequence of indices $\nu_j + 1 = a_1 < a_2 < \dots < a_t \leq \nu_{j+1} - 1$ ($\nu_{r+1} = n$ and $\nu_0 = -1$) such that

- the transition $I_{a_{i+1}-1} \mapsto I_{a_i}$ is an \mapsto_{atmost} transition ($1 \leq i \leq t - 1$)
- the computation $I_{a_i} \mapsto \dots \mapsto I_{a_{i+1}-1}$ is a *AL-pure* computation.

We illustrate this definition in the next example.

Example 18

Consider the program of Example 10. A possible computation of M_1 is:³

$$\begin{array}{ccccccc} \langle \emptyset, \emptyset \rangle & \mapsto_{AL^1} & \langle \{e\}, \emptyset \rangle & \mapsto_{AL^1} & \langle \{e, f\}, \emptyset \rangle & \mapsto_{atmost} & \\ \langle \{e, f\}, \{c, d\} \rangle & \mapsto_{choice} & \langle \{e, f, b\}, \{c, d\} \rangle & \mapsto_{AL^2} & \langle \{e, f, b\}, \{c, d, a\} \rangle & & \end{array}$$

□

6.2 SMOBELES *On-line Justifications*

We can use knowledge of the specific steps performed by SMOBELES to guide the construction of an on-line justification.

Assuming that

$$C = M_0 \mapsto M_1 \mapsto M_2 \mapsto \dots \mapsto M_n$$

is a computation of SMOBELES. Let $S(M_i) = \langle E_1, E_2, D \rangle$ and $S(M_{i+1}) = \langle E'_1, E'_2, D' \rangle$ be the snapshots correspond to M_i and M_{i+1} respectively. Obviously, $S(M_{i+1})$ can be computed by the following steps:

- computing $D' = \langle \Gamma(M_{i+1}), \Delta(M_{i+1}) \rangle$;
- updating E_1 and E_2 to obtain E'_1 and E'_2 .

We observe that $\langle \Gamma(M_{i+1}), \Delta(M_{i+1}) \rangle$ can be obtained by computing the fixpoint of the Γ - and Δ -function with the starting value $\Gamma_{\langle \Gamma(M_i), \Delta(M_i) \rangle}$ and $\Delta_{\langle \Gamma(M_i), \Delta(M_i) \rangle}$. This is possible due to the monotonicity of the computation. Regarding E'_1 and E'_2 ,

³ We omit the steps that do not change the interpretation.

observe that the e-graphs for elements in $\langle \Gamma^k(M_{i+1}), \Delta^k(M_{i+1}) \rangle$ can be constructed using the e-graphs constructed for elements in $\langle \Gamma^{k-1}(M_{i+1}), \Delta^{k-1}(M_{i+1}) \rangle$ and the rules involved in the computation of $\langle \Gamma^k(M_{i+1}), \Delta^k(M_{i+1}) \rangle$. Thus, we only need to update E'_1 with e-graphs of elements of $\langle \Gamma^k(M_{i+1}), \Delta^k(M_{i+1}) \rangle$ which do not belong to $\langle \Gamma^{k-1}(M_{i+1}), \Delta^{k-1}(M_{i+1}) \rangle$. Also, E'_2 is obtained from E_2 by removing the e-graphs of atoms that “move” into D' and adding the e-graph $(a, \text{assume}, +)$ (resp. $(a, \text{assume}, -)$) for $a \in M_{i+1}^+$ (resp. $a \in M_{i+1}^-$) not belonging to D' . Clearly, this computation depends on the transition from M_i to M_{i+1} . Assume that $M_i \mapsto_\alpha M_{i+1}$, the update of $S(M_i)$ to create $S(M_{i+1})$ is done as follows.

- $\boxed{\alpha \equiv \text{choice:}}$ let p be the atom chosen in this step.
If p is chosen to be true, then we can use the graph

$$G_p = (\{a, \text{assume}\}, \{(a, \text{assume}, +)\})$$

and the resulting snapshot is $S(M_{i+1}) = \langle E_1, E_2 \cup \{G_p\}, D \rangle$. Observe that D is unchanged, since the structure of the computation (in particular the fact that an *expand* has been done before the choice) ensures that p will not appear in the computation of D .

If p is chosen to be false, then we will need to add p to D^- , compute $\Gamma(M_{i+1})$ and $\Delta(M_{i+1})$, and update E_1 and E_2 correspondingly; in particular, p belongs to $\Delta(M_{i+1})$ and $G_p = (\{a, \text{assume}\}, \{(a, \text{assume}, -)\})$ is added to E_1 .

- $\boxed{\alpha \equiv \text{atmost:}}$ in this case, $M_{i+1} = \langle M_i^+, M_i^- \cup \text{AtMost}(P, M_i) \rangle$. The computation of $S(M_{i+1})$ is performed as from definition of on-line justification. In particular, observe that if $\forall c \in \text{AtMost}(P, M_i)$ we have that $LCE_P^n(c, D) \neq \emptyset$ then the computation can be started from $\Gamma(M_i)$ and $\Delta(M_i) \cup \text{AtMost}(P, M_i)$.
- $\boxed{\alpha \equiv \text{AL}^1:}$ let p be the atom dealt with in this step and let r be the rule employed. We have that $M_{i+1} = \langle M_i^+ \cup \{p\}, M_i^- \rangle$. If $D \models \text{body}(r)$ then $S(M_{i+1})$ will be computed starting from $\Gamma(M_i) \cup \{p\}$ and $\Delta(M_i)$. In particular, an off-line graph for p , let's say G_p , will be added to E_1 , and such graph will be constructed using the LCE based on the rule r and the e-graphs in E_1 .
Otherwise, $S(M_{i+1}) = \langle E_1, E_2 \cup \{G^+(p, r, \Sigma)\}, D \rangle$, where $G^+(p, r, \Sigma)$ is an e-graph of p^+ constructed using the LCE of rule r and the e-graphs in $\Sigma = E_1 \cup E_2$ (note that all elements in $\text{body}(r)$ have an e-graph in $E_1 \cup E_2$).
- $\boxed{\alpha \equiv \text{AL}^2:}$ let p be the atom dealt with in this step. In this case $M_{i+1} = \langle M_i^+, M_i^- \cup \{p\} \rangle$. If there exists $\gamma \in LCE_P^n(p, D, \emptyset)$, then $S(M_{i+1})$ can be computed according to the definition of on-line justification, starting from $\Gamma(M_i)$ and $\Delta(M_i) \cup \{p\}$. Observe that the graph of p can be constructed starting with $\{(p, a, +) \mid a \in \gamma\} \cup \{(p, b, -) \mid \text{not } b \in \gamma\}$.
Otherwise, given an arbitrary $\psi \in LCE_P^n(p, M_i, \emptyset)$, we can construct an e-graph G_p for p^- , such that $\psi = \text{support}(p^-, G_p)$, the graphs $E_1 \cup E_2$ are used to describe the elements of ψ , and $S(M_{i+1}) = \langle E_1, E_2 \cup \{G_p\}, D \rangle$.
- $\boxed{\alpha \equiv \text{AL}^3:}$ let r be the rule used in this step and let $p = \text{head}(r)$. Then $M_{i+1} = \langle M_i^+ \cup \text{pos}(r), M_i^- \cup \text{neg}(r) \rangle$ and $S(M_{i+1})$ is computed according to the definition of on-line justification. Observe that the e-graph G_p for p^+ (added to E_1 or E_2) for $S(M_{i+1})$ will be constructed using $\text{body}(r)$ as

$support(p^+, G_p)$, and using the e-graphs in $E_1 \cup E_2 \cup \Sigma$ for some

$$\Sigma \subseteq \{(a^+, assume, +) \mid a \in pos(r)\} \cup \{(a, assume, -) \mid a \in neg(r)\}.$$

- $\boxed{\alpha \equiv AL^4}$: let r be the rule processed and let b the atom detected in the body. If $b \in pos(r)$, then $M_{i+1} = \langle M_i^+, M_i^- \cup \{b\} \rangle$, while if $b \in neg(r)$ then $M_{i+1} = \langle M_i^+ \cup \{b\}, M_i^- \rangle$. In either cases, the snapshot $S(M_{i+1})$ will be computed using the definition of on-line justification.

Example 19

Let us consider the computation of Example 18. A sequence of snapshots is (we provide only the edges of the graphs and we combine together e-graphs of different atoms):

	E_1	E_2	D
$S(M_0)$	\emptyset	\emptyset	\emptyset
$S(M_1)$	$\{(e^+, \top, +)\}$	\emptyset	$\langle \{e\}, \emptyset \rangle$
$S(M_2)$	$\{(e^+, \top, +), (f^+, e^+, +)\}$	\emptyset	$\langle \{e, f\}, \emptyset \rangle$
$S(M_3)$	$\left\{ \begin{array}{l} (e^+, \top, +), (f^+, e^+, +) \\ (d^-, c^-, +), (c^-, d^-, +) \end{array} \right\}$	\emptyset	$\langle \{e, f\}, \{c, d\} \rangle$
$S(M_4)$	$\left\{ \begin{array}{l} (e^+, \top, +), (f^+, e^+, +) \\ (d^-, c^-, +), (c^-, d^-, +) \end{array} \right\}$	$\{(b^+, assume, +)\}$	$\langle \{e, f\}, \{c, d\} \rangle$
$S(M_5)$	$\left\{ \begin{array}{l} (e^+, \top, +), (f^+, e^+, +), \\ (d^-, c^-, +), (c^-, d^-, +), \\ (a^-, assume, -), \\ (b^+, e^+, +), (b^+, a^-, -) \end{array} \right\}$	\emptyset	$\langle \{e, f, b\}, \{c, d, a\} \rangle$

□

Example 20

Let P be the program:

$$\begin{array}{ll} p & :- \text{ not } q & q & :- \text{ not } p \\ r & :- \text{ not } p & p & :- r \end{array}$$

This program does not admit any answer sets where p is false. One possible computation (we highlight only steps that change the trace):

1. $\langle \emptyset, \emptyset \rangle \mapsto_{choice}$
2. $\langle \emptyset, \{p\} \rangle \mapsto_{AL^1}$
3. $\langle \{q\}, \{p\} \rangle \mapsto_{AL^1}$
4. $\langle \{q, r\}, \{p\} \rangle \mapsto_{AL^1}$
5. $\langle \{q, r, p\}, \{p\} \rangle$

From this computation we can obtain a sequence of snapshots:

	E_1	E_2	D
$S(M_0)$	\emptyset	\emptyset	\emptyset
$S(M_1)$	$\{(p^-, \text{assume}, -)\}$	\emptyset	$\langle \emptyset, \{p\} \rangle$
$S(M_2)$	$\{(p^-, \text{assume}, -), (q^+, p^-, -)\}$	\emptyset	$\langle \{q\}, \{p\} \rangle$
$S(M_3)$	$\{(p^-, \text{assume}, -), (q^+, p^-, -), (r^+, p^-, -)\}$	\emptyset	$\langle \{q, r\}, \{p\} \rangle$
$S(M_4)$	$\left\{ \begin{array}{l} (p^-, \text{assume}, -), (q^+, p^-, -), \\ (r^+, p^-, -), (p^+, r^+, +) \end{array} \right\}$	\emptyset	$\langle \{p, q, r\}, \{p\} \rangle$

Observe that a conflict is detected by the computation and the sources of conflict are highlighted in the presence of two justifications for p , one for p^+ and another one for p^- . \square

6.3 Discussion

In this subsection, we discuss possible ways to extend the notion of justifications on various language extensions of ASP. We also describe a system capable of computing off-line and on-line justifications for ASP programs.

6.3.1 Language Extensions

In the discussion presented above, we relied on a standard logic programming language. Various systems, such as SMOBELS, have introduced language extensions, such as choice atoms, to facilitate program development. The extension of the notion of justification to address these extensions is relatively straightforward.

Let us consider, for example, the choice atom construct of SMOBELS. A choice atom has the form $L \leq \{a_1, \dots, a_n, \text{not } b_1, \dots, \text{not } b_m\} \leq U$ where L, U are integers (with $L \leq U$) and the various a_i, b_j are atoms. Choice atoms are allowed to appear both in the head as well as the body of rules. Given an interpretation I and a choice atom, we say that I satisfies the atom if

$$L \leq |\{a_i \mid a_i \in I^+\}| + |\{b_j \mid b_j \in I^-\}| \leq U$$

The local consistent explanation of a choice atom can be developed in a natural way:

- If the choice atom $L \leq T \leq U$ is true, then a set of literals S is an LCE if
 - $\mathcal{A} \cap S \subseteq T$ and $\text{not } \mathcal{A} \cap S \subseteq T$
 - for each S' such that $S \subseteq S'$ and $\{\text{atom}(\ell) \mid \ell \in S'\} = \{\text{atom}(\ell) \mid \ell \in T\}$ we have that

$$L \leq |\{a \mid a \in T \cap \mathcal{A} \cap S'\}| + |\{b \mid \text{not } b \in T \cap S'\}| \leq U$$

- if the choice atom $L \leq T \leq U$ is false, then a set of literals S is an LCE if

- $\mathcal{A} \cap S \subseteq T$ and *not* $\mathcal{A} \cap S \subseteq T$
- for each S' such that $S \subseteq S'$ and $\{atom(\ell) \mid \ell \in S'\} = \{atom(\ell) \mid \ell \in T\}$ we have that

$$L > |\{a \mid a \in T \cap \mathcal{A} \cap S'\}| + |\{b \mid not\ b \in T \cap S'\}|$$

or

$$|\{a \mid a \in T \cap \mathcal{A} \cap S'\}| + |\{b \mid not\ b \in T \cap S'\}| > U$$

The notions of e-graphs can be extended to include choice atoms. Choice atoms in the body are treated as such and justified according to the new notion of LCE. On the other hand, if we have a rule of the type

$$L \leq T \leq U \text{ :- } Body$$

and M is an answer set, then we will

- treat the head as a new (non-choice) atom ($new_{L \leq T \leq U}$), and allow its justification in the usual manner, using the body of the rule
- for each atom $p \in T \cap M^+$, the element p^+ has a new LCE $\{new_{L \leq T \leq U}\}$

Example 21

Consider the program containing the rules:

$$\begin{array}{l}
 p \text{ :-} \qquad \qquad \qquad q \text{ :-} \\
 2 \leq \{r, t, s\} \leq 2 \text{ :- } p, q
 \end{array}$$

The interpretation $\langle \{t, s, p, q\}, \{r\} \rangle$ is an answer set of this program. The off-line justifications for s^+ and t^+ are illustrated in Figure 7. □

The concept can be easily extended to deal with weight atoms.

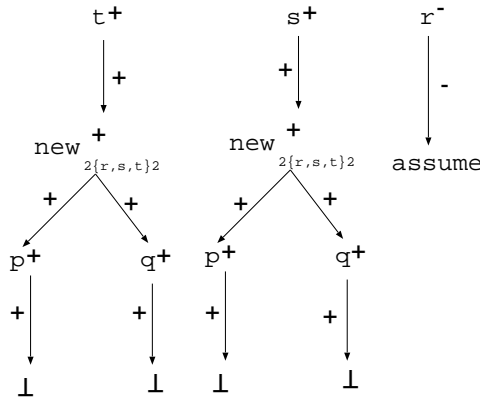


Fig. 7. Justifications in presence of choice atoms

6.3.2 Concrete Implementation

The notions of off-line and on-line justifications proposed in the previous sections have been implemented and integrated in a debugging system for Answer Set Programming, developed within the ASP – PROLOG framework (Elkhatib et al. 2004). The notions of justification proposed here is meant to represent the basic data structure on which debugging strategies for ASP can be developed. ASP – PROLOG allows the construction of Prolog programs—using CIAO Prolog (Gras and Hermenegildo 2000)—which include modules written in ASP (the SMOELS flavor of ASP). In this sense, the embedding of ASP within a Prolog framework (as possible in ASP – PROLOG) allows the programmer to use Prolog itself to query the justifications and develop debugging strategies. We will begin this section with a short description of the system ASP – PROLOG.

The ASP – PROLOG system has been developed using the module and class capabilities of CIAO Prolog. ASP – PROLOG allows programmers to develop programs as collections of *modules*. Along with the traditional types of modules supported by CIAO Prolog (e.g., Prolog modules, Constraint Logic Programming modules), it allows the presence of *ASP modules*, each being a complete ASP program. Each CIAO Prolog module can access the content of any ASP module (using the traditional module qualification of Prolog), read its content, access its models, and modify it (using the traditional `assert` and `retract` predicates of Prolog).

Example 22

ASP – PROLOG allows us to create Prolog modules that access (and possibly modify) other modules containing ASP code. For example, the following Prolog module

```
:- use_asp(aspmod, 'asp_module.lp').

count_p(X) :-
    findall(Q, (aspmod:model(Q), Q:p), List),
    length(List,X).
```

accesses an ASP module (called `aspmod`) and defines a predicate (`count_p`) which counts how many answer sets of `aspmod` contain the atom `p`. □

Off-Line Justifications: The SMOELS engine has been modified to extract, during the computation, a compact footprint of the execution, i.e., a trace of the key events (corresponding to the transitions described in Sect. 6) with links to the atoms and rules involved. The modifications of the trace are trailed to support backtracking. Parts of the justification (as described in the previous section) are built on the fly, while others (e.g., certain cases of AL^3 and AL^4) are delayed until the justification is requested.

To avoid imposing the overhead of justification construction on every computation, the programmer has to specify what ASP modules require justifications, using an additional argument (`justify`) in the module import declaration:

```
:- use_asp(< module_name >, < file_name >, < parameters > [,justify]).
```


Figure 8 shows a general overview of the implementation of ASP justifications in ASP – PROLOG. Each program is composed of CIAO Prolog modules and ASP modules (each containing rules of the form (1), possibly depending on the content of other ASP/Prolog modules). The implementation of ASP – PROLOG, as described in (Elkhatib et al. 2004), automatically generates, for each ASP module, an *interface module*—which supplies the predicates to access/modify the ASP module and its answer sets—and a *model class*—which allows the encoding of each answer set as a CIAO Prolog object (Pineda 1999). The novelty is the extension of the model class, to provide access to the justification of the elements in the corresponding answer set.

ASP – PROLOG provides the predicate `model/1` to retrieve answer sets of an ASP module—it retrieves them in the order they are computed by `S MODELS`, and it returns the current one if the computation is still in progress. The main predicate to access the justification is `justify/1` which retrieves a CIAO Prolog object containing the justification; i.e.,

```
?- my_asp:model(Q), Q:justify(J).
```

will assign to `J` the object containing the justification relative to the answer set `Q` of the ASP module `my_asp`. Each justification object provides the following predicates:

- `just_node/1` which succeeds if the argument is one of the nodes in the justification graph,
- `just_edge/3` which succeeds if the arguments correspond to the components of one of the edges in the graph, and
- `justify_draw/1` which will generate a graphical drawing of the justification for the given atom (using the `uDrawGraph` application). An example display produced by ASP – PROLOG is shown in Figure 9; observe that rule names are also displayed to clarify the connection between edges of a justification and the generating program rules.

For example,

```
?- my_asp:model(Q),Q:justify(J),findall(e(X,Y),J:just_edge(p,X,Y),L).
```

will collect in `L` all the edges supporting `p` in the justification graph (for answer set `Q`).

On-Line Justifications: The description of `S MODELS` on-line justifications we proposed earlier is clearly more abstract than the concrete implementation—e.g., we did not address the use of lookahead, the use of heuristics, and other optimizations introduced in `S MODELS`. All these elements have been handled in the current implementation, in the same spirit of what described here.

On-line justifications have been integrated in the ASP – PROLOG system as part of its ASP debugging facilities. The system provides predicates to set breakpoints on the execution of an ASP module, triggered by events such as the assignment of a truth value to a certain atom or the creation of a conflicting assignment. Once a

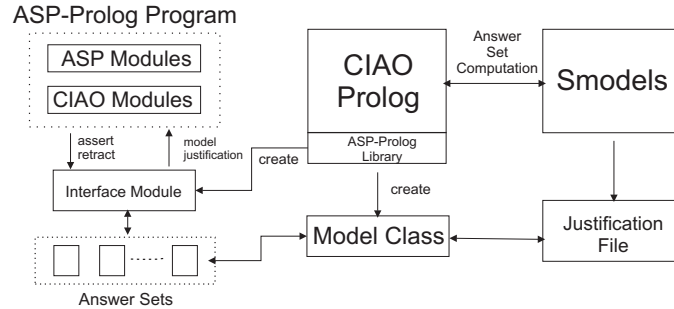


Fig. 8. ASP – PROLOG with justifications

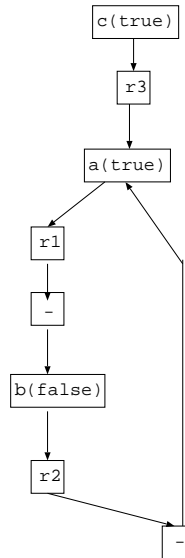


Fig. 9. An off-line justification produced by ASP – PROLOG

breakpoint is encountered, it is possible to visualize the current on-line justification or step through the rest of the execution. Off-line justifications are always available.

The SMOBELS solver is in charge of handling the activities of interrupting and resuming execution, during the computation of an answer set of an ASP program. A synchronous communication is maintained between a Prolog module and an ASP module—where the Prolog module requests and controls the ASP execution. When the ASP solver breaks, e.g., because a breakpoint is encountered, it sends a compact encoding of its internal data structures to the Prolog module, which stores it in a ASP-solver-state table. If the Prolog module requests resumption of the ASP execution, it will send back to the solver the desired internal state, that will allow continuation of the execution. This allows the execution to be restarted from any of a number of desired points (e.g., allowing a “replay”-style of debugging) and to control different ASP modules at the same time.

ASP – PROLOG provides the ability to establish a number of different types of breakpoints on the execution of an ASP module. In particular,

- `break(atom,value)` interrupts the execution when the `atom` is assigned the given value; `value` could `true`, `false` or `any`.
- `break(conflict)` interrupts the execution whenever a conflict is encountered during answer set computation.⁴
- `break(conflict(atom))` interrupts the execution if a conflict involving the `atom` is encountered.
- `break(answer(N))` interrupts the execution at the end of the computation of the answer set referred to by the object `N`.

Execution can be restarted using the built-in predicate `run`; the partial results of an interrupted computation (e.g., the partial answer set, the on-line justification) can be accessed using the predicates `model` and `justify`.

Example 23

Consider the following fragment of a Prolog program:

```
:- module ( p, [m/O] ).
:- use_asp ( asp, 'myasp.lp', justify ).

m :- asp:break(atom(a,true)),
     asp:run,
     asp:model(Q),
     Q:justify(J),
     J:justify_draw(a).
```

This will stop the execution of the answer set program `myasp.lp` whenever the atom `a` is made true; at that point, the Prolog program shows a graphical representation of the corresponding on-line justification of `a`. □

6.4 Justifications and Possible Applications

The previous subsection discusses a possible application of the notion of justification developed in this paper, namely the construction of an interactive debugging system for logic programs under the answer set semantics. It is worth mentioning that the notion of justification is general and can be employed in other applications as well. We will now briefly discuss other potential uses of this concept.

Thanks to their ability to explain the presence and absence of atoms in an answer set, off-line justifications provide a natural solution to problems in the domain of ASP-based diagnosis. As in systems like (Balduccini and Gelfond 2003), off-line justifications can help in discriminating diagnoses. Let us consider, for example, a system composed of two components, c_1 and c_2 . Let us assume that there is a dependence between these components, stating that if c_1 is defective then c_2 will be defective as well. This information can be expressed by the following rule:

$$h(ab(c_2), T) \text{ :- } h(ab(c_1), T)$$

⁴ Here, we refer to conflict in the same terms as `S MODELS`.

where $h(ab(c_1), t)$ (resp. $h(ab(c_2), t)$) being true indicates that the component c_1 (resp. c_2) is defective at an arbitrary time T .

Given this rule, $h(ab(c_2), t)$ ($ab(c_2)$ is defective) belongs to any answer set which contain $h(ab(c_1), t)$ ($ab(c_1)$ is defective). Thus, any off-line justification for $h(ab(c_1), t)^+$ can be extended to an off-line justification for $h(ab(c_2), t)^+$ by adding a positive edge from $h(ab(c_2), t)^+$ to $h(ab(c_1), t)^+$. This is another argument, besides the minimality criteria, for preferring the diagnosis $\{c_1\}$ over $\{c_1, c_2\}$.

The implemented system for on-line justification in this paper can be adapted to create a direct implementation of the CR-Prolog (Balduccini 2007). Currently, a generate-and-test front-end to SMOBELS is provided for computing answer sets of CR-Prolog programs. More precisely, the algorithm for computing the answer sets of a CR-Prolog program P , whose set of normal rules is Q , iterates through two steps until an answer set is found. In each iteration, a minimal set of CR-rules is selected randomly (or according to some preferences), activated (i.e., converted to normal rules) and added to Q to create a new program Q' . The answer sets of Q' are computed using SMOBELS. If any answer set is found, then the computation stops.

This implementation does not make use of any information about possible conflicts or inconsistencies that can be recognized during the computation. A more effective implementation can be achieved by collecting on-line justifications during each cycle of execution of SMOBELS. The on-line justifications can be traversed to identify inconsistencies and identify rules outside of Q that unavoidably conflict with rules in Q . Such knowledge can then be employed to suggest more effective selections of CR-rules to be activated.

Example 24

Consider the following simple CR-Prolog program

$$\begin{array}{llll} r_1 & a & :- & not\ b. \\ r_2 & \neg a & & \\ r_3 & b & \leftarrow^+ & \\ r_4 & c & \leftarrow^+ & \end{array}$$

In this case, the set of normal rules Q contains the two rules r_1 and r_2 , and Q does not admit a (consistent) answer set. The point of conflict is characterized by the on-line justification shown in Figure 10. The conflict is clearly represented by the presence of justifications for a^+ and $(\neg a)^+$; the justification also highlights that the only way of defeating the conflict is to remove the positive edge between $not\ b$ and a^+ . This suggests the need of introducing a CR-rule that has b as head, i.e., rule r_3 .

Simple ASP – PROLOG meta-interpreters can be introduced to detect this type of situations and suggest some consistency restoring rules to be used; e.g., given the partial answer set M present at the time of conflict, we can use the following

clause to resolve conflicts due to atoms of the type p and $\neg p$ both being true:

```

candidate_rule( $Y \overset{\pm}{\leftarrow} Body, M$ ) :-
     $M : justify(J)$ ,
     $M : Atom, M : (\neg Atom)$ ,
    ( $reachable(Atom, Y, J); reachable(\neg Atom, Y, J)$ ),
     $M : not Y$ ,
     $\overset{\pm}{\leftarrow} (Y, Body)$ .
    
```

where `reachable` performs a simple transitive closure over the edges of the justification J .

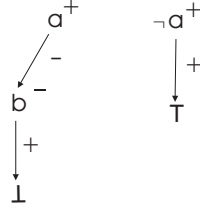


Fig. 10. On-line justification for the rules r_1 and r_2

□

7 Related Work

Various approaches to logic program understanding and debugging have been investigated (and a thorough comparison is beyond the limited space of this paper). Early work in this direction geared towards the understanding of Prolog programs rather than logic programs under the answer set semantics. Only recently, we can find some work on debugging inconsistent programs or providing explanation for the presence (or absence) of an atom in an answer set. While our notion of justification is related to the research aimed at debugging Prolog and XSB programs, its initial implementation is related to the recent attempts in debugging logic programs under the answer set semantics. We will discuss each of these issues in each subsection.

7.1 Justifications and Debugging of Prolog Programs

As discussed in (Pemasani et al. 2004), 3 main phases can be considered in understanding/debugging a logic program:

1. *Program instrumentation and execution*: assertion-based debugging (e.g., (Puebla et al. 1998)) and algorithmic debugging (Shapiro 1982) are examples of approaches focused on this first phase.
2. *Data Collection*: focuses on *extracting* from the execution data necessary to understand it, as in event-based debugging (Auguston 2000), tracing, and explanation-based debugging (Ducassé 1999; Mallet and Ducassé 1999).

3. *Data Analysis*: focuses on reasoning on data collected during the execution. The proposals dealing with automated debugging (e.g., (Auguston 2000)) and execution visualization (e.g., (Vaupel et al. 1997)) are approaches focusing on this phase of program understanding.

The notion of *Justification* has been introduced in (Pemmasani et al. 2004; Roychoudhury et al. 2000; Specht 1993) to support understanding and debugging of logic programs. Justification is the process of generating evidence, in terms of high-level proofs based on the answers (or models) produced during the computation. Justifications are *focused*, i.e., they provide only the information that are relevant to the item being explained—and this separates them from other debugging schemes (e.g., tracing). Justification plays an important role in manual and automatic verification, by providing a *proof description* if a given property holds; otherwise, it generates a *counter-example*, showing where the violation/conflict occurs in the system. The justification-based approach focuses on the last two phases of debugging—collecting data from the execution and presenting them in a meaningful manner. Differently from generic tracing and algorithmic debugging, justifications are focused only on parts of the computation relevant to the justified item. Justifications are fully automated and do not require user interaction (as in declarative debugging).

Justifications relies on describing the evidence for an answer in terms of a graph structure. The term *justification* was introduced in (Roychoudhury et al. 2000), as a data structure to explain answers to Prolog queries within a Prolog system with tabling. The notion of justification and its implementation in the XSB Prolog system was successively refined in (Pemmasani et al. 2004; Guo et al. 2001). Similar structures have been suggested to address the needs of other flavors of logic programming—e.g., various approaches to tree-based explanation for deductive databases (e.g., the *Explain* system (Arora et al. 1993), the explanation system for LOLA (Specht 1993), and the DDB trees method (Mallet and Ducassé 1999)). Similar methods have also been developed for the analysis of CLP programs (e.g., (Deransart et al. 2000)).

In this work, we rely on graph structures as a mean to describe the *justifications* that are generated during the generation (or from) an answer set of a program. Graphs have been used in the context of logic programming for a variety of other applications. *Call graphs* and *dependence graphs* have been extensively used to profile and discover program properties (e.g., (Mera et al. 2006; Debray et al. 1997)). *Support graphs* are used for program analysis in (Saha and Ramakrishnan 2005).

The use of graphs proposed in this paper is complementary to the view proposed by other authors, who use graph structures as a mean to describe answer set programs, to make structural properties explicit, and to support the development of the program execution. In (Anger et al. 2005; Konczak et al. 2006), *rule dependency graphs* (a.k.a. *block graphs*) of answer set programs are employed to model the computation of answer sets as special forms of graph coloring. A comprehensive survey of alternative graph representations of answer set programs, and their properties with respect to the problem of answer set characterization, has been

presented in (Costantini 2001; Costantini et al. 2002). In particular, the authors provide characterizations of desirable graph representations, relating the existence of answer sets to the presence of cycles and the use of coloring to characterize properties of programs (e.g., consistency). We conjecture that the outcome of a successful coloring of an EDG (Costantini 2001) to represent one answer set can be projected, modulo non-obvious transformations, to an off-line graph and vice versa. On the other hand, the notion of on-line justification does not seem to have a direct relation to the graph representations presented in the cited works.

7.2 Debugging Logic Programs under Answer Set Semantics

This paper continues the work initiated in (El-Khatib et al. 2005), by proposing a more advanced and sound notion of off-line justification, by developing the concept of on-line justification, and introducing these concepts in SMOBELS. The approach differs significantly from the recently introduced approach to debugging ASP programs in (Brain et al. 2007a). While our approach relies on the notion of justification, the approach in (Brain et al. 2007a) uses the tagging technique (Delgrande et al. 2003) to compile a program into a new program whose answer sets can be used to debug the original program. Inspecting an answer set of the new program can reveal the rules which have been applied in its generation. It does not, however, provide explanation of why an atom does (or does not) belong to the answer set. In this sense, we can say that the approach of (Brain et al. 2007a) and ours are complementary to each other. An advantage of the approach in (Brain et al. 2007a) is that it enables the development of a debugger as a front-end of an answer set solver. However, their approach does not consider on-line justification.

At this point, it is worth mentioning that the ASP – PROLOG debugger, described in Section 6, differs from the system `spock` (Brain et al. 2007b)—which was developed based on the technical foundation in (Brain et al. 2007a)—in several aspects. In our system, the justification for the truth value of an atom consists of facts, assumptions, and rules which are applicable given these facts and assumptions, i.e., we not only justify why an atom is *true* but also why an atom is *false*. Moreover, justifications can be queried during the process of answer set computation. `spock` only provides the justification, or the applicable rules, for the presence of an atom in a given answer set. In this sense, justifications in `spock` is similar to our off-line LCEs.

In (Perri et al. 2007), a tool for developing and testing DLV programs was described. The commands provided by this tool allow an user to inspect why an atom is true in the current model and why there is no answer set. This is similar to the on-line justifications developed for SMOBELS. The tool in (Perri et al. 2007), however, does not answer the question why an atom is not in the current model. The notion of justifications is not developed in (Perri et al. 2007).

The proposed debugger is similar to the system described in (Brain and de Vos 2005) in that it provides the users with the information on why some atoms occur in an answer set and some others do not. An explanation given by the tool described in this work is similar to an off-line justification in our work. Our implementation

also provides users with on-line justifications but the system described in (Brain and de Vos 2005) does not.

The paper (Syrjänen 2006) presents a theory for debugging of inconsistent programs and an implementation of this theory. The focus of this paper is on inconsistent programs. On the other hand, our focus is not solely on inconsistent programs. Our notion of on-line justification can be used in identifying the reasons that lead to the inconsistency of the problem but it is significant different from the theory of diagnosis developed in (Syrjänen 2006).

8 Conclusion

In this paper we provided a generalization of the notion of *justification* (originally designed for Prolog with SLG-resolution (Roychoudhury et al. 2000)), to suit the needs of ASP. The notion, named *off-line justification*, offers a way to understand the motivations for the truth value of an atom within a specific answer set, thus making it easy to analyze answer sets for program understanding and debugging. We also introduced *on-line justifications*, which are meant to justify atoms *during* the computation of an answer set. The structure of an on-line justification is tied to the specific steps performed by a computational model for ASP (specifically, the computation model adopted by SMOBELS). An on-line justification allows a programmer to inspect the reasons for the truth value of an atom at the moment such value is determined while constructing an answer set. These data structures provide a foundation for the construction of tools to understand and debug ASP programs.

The process of computing and presenting justifications has been embedded in the ASP-Prolog system (Elkhatib et al. 2004), thus making justifications a first-class citizen of the language. This allows the programmer to use Prolog to manipulate justifications as standard Prolog terms. A prototype implementation has been completed and is currently under testing.

As future work, we propose to complete the implementation, refine the definition of on-line justification to better take advantage of the SMOBELS mechanisms, and develop a complete debugging and visualization environment for ASP based on these data structures.

Acknowledgement: We would like to thank the anonymous reviewers for their comments and suggestions that help improve the papers in many ways. The authors are partially supported by NSF grants CNS-0220590, HRD-0420407, and IIS-0812267.

References

- ANGER, C., GEBSER, M., LINKE, T., NEUMANN, A., AND SCHAUB, T. 2005. The nomore++ approach to answer set solving. In *Proceedings of the 12th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*. 95–109.
- APT, K. AND BOL, R. 1994. Logic programming and negation: a survey. *Journal of Logic Programming* 19,20, 9–71.

- ARORA, T., RAMAKRISHNAN, R., ROTH, W., SESHADRI, P., AND DRIVASTAVA, D. 1993. Explaining Program Execution in Deductive Systems. In *Proceedings of the DOOD Conference*. Springer Verlag.
- AUGUSTON, M. 2000. Assertion checker for the C programming language based on computations over event traces. In *AADEBUG*.
- BALDUCCINI, M. 2007. cr-models: An inference engine for cr-prolog. In *LPNMR*, C. Baral, G. Brewka, and J. S. Schlipf, Eds. Lecture Notes in Computer Science, vol. 4483. Springer, 18–30.
- BALDUCCINI, M. AND GELFOND, M. 2003. Diagnostic Reasoning with A-Prolog. *Theory and Practice of Logic Programming* 3, 4,5, 425–461.
- BALDUCCINI, M., GELFOND, M., AND NOGUEIRA, M. 2006. Answer Set Based Design of Knowledge Systems. *Annals of Mathematics and Artificial Intelligence*.
- BARAL, C. 2003. *Knowledge Representation, reasoning, and declarative problem solving with Answer sets*. Cambridge University Press, Cambridge, MA.
- BRAIN, M. AND DE VOS, M. 2005. Debugging Logic Programs under the Answer Set Semantics. In *Answer Set Programming: Advances in Theory and Implementation*, M. D. Vos and A. Provetti, Eds. 142–152.
- BRAIN, M., GEBSER, M., PÜHRER, J., SCHAUB, T., TOMPITS, H., AND WOLTRAN, S. 2007a. Debugging ASP programs by means of ASP. In *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, C. Baral, G. Brewka, and J. Schlipf, Eds. Lecture Notes in Artificial Intelligence, vol. 4483. Springer-Verlag, 31–43.
- BRAIN, M., GEBSER, M., PÜHRER, J., SCHAUB, T., TOMPITS, H., AND WOLTRAN, S. 2007b. “That is illogical captain!” – The debugging support tool spock for answer-set programs: System description. In *Proceedings of the Workshop on Software Engineering for Answer Set Programming (SEA'07)*, M. De Vos and T. Schaub, Eds. 71–85.
- BRASS, S., DIX, J., FREITAG, B., AND ZUKOWSKI, U. 2001. Transformation-based bottom-up computation of the well-founded model. *TPLP* 1, 5, 497–538.
- COSTANTINI, S. 2001. Comparing Different Graph Representations of Logic Programs under the Answer Set Semantics. In *Answer Set Programming Workshop*.
- COSTANTINI, S., D’ANTONA, O. M., AND PROVETTI, A. 2002. On the equivalence and range of applicability of graph-based representations of logic programs. *Inf. Process. Lett.* 84, 5, 241–249.
- DAVIS, M., LOGEMANN, G., AND LOVELAND, D. W. 1962. A machine program for theorem-proving. *Commun. ACM* 5, 7, 394–397.
- DEBRAY, S., LOPEZ-GARCIA, P., HERMENEGILDO, M., AND LIN, N. 1997. Lower Bound Cost Estimation for Logic Programs. In *International Logic Programming Symposium*. MIT Press, 291–305.
- DELGRANDE, J., SCHAUB, T., AND TOMPITS, H. 2003. A framework for compiling preferences in logic programs. *Theory and Practice of Logic Programming* 3, 2 (Mar.), 129–187.
- DERANSART, P., HERMENEGILDO, M. V., AND MALUSZYNSKI, J., Eds. 2000. *Analysis and Visualization Tools for Constraint Programming, Constraint Debugging (DiSCiPl project)*. Lecture Notes in Computer Science, vol. 1870. Springer.
- DUCASSÉ, M. 1999. Opium: An extendable trace analyzer for prolog. *J. Log. Program.* 39, 1-3, 177–223.
- EITER, T., LEONE, N., MATEIS, C., PFEIFER, G., AND SCARCELLO, F. 1998. The KR System DLV: Progress Report, Comparisons, and Benchmarks. In *International Conference on Principles of Knowledge Representation and Reasoning*. 406–417.

- EL-KHATIB, O., PONTELLI, E., AND SON, T. C. 2005. Justification and debugging of answer set programs in ASP. In *Proceedings of the Sixth International Workshop on Automated Debugging, AADEBUG 2005, Monterey, California, USA, September 19-21, 2005*, C. Jeffery, J.-D. Choi, and R. Lencevicius, Eds. ACM, 49–58.
- ELKHATIB, O., PONTELLI, E., AND SON, T. 2004. ASP-Prolog: A System for Reasoning about Answer Set Programs in Prolog. In *Proceedings of the Sixth International Symposium on Practical Aspects of Declarative Languages (PADL-2004)*. Springer, 148–162.
- ERDEM, E., LIFSCHITZ, V., AND RINGE, D. 2006. Temporal phylogenetic networks and logic programming. *TPLP* 6, 5, 539–558.
- FAGES, F. 1994. Consistency of Clark’s completion and existence of stable models. *Methods of Logic in Computer Science* 1, 51–60.
- GEBSER, M., KAUFMANN, B., NEUMANN, A., AND SCHAUB, T. 2007. clasp: A conflict-driven answer set solver. In *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR’07)*, C. Baral, G. Brewka, and J. Schlipf, Eds. Lecture Notes in Artificial Intelligence, vol. 4483. Springer-Verlag, 260–265.
- GEBSER, M., SCHAUB, T., AND THIELE, S. 2007. Gringo: A new grounder for answer set programming. In *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR’07)*, C. Baral, G. Brewka, and J. Schlipf, Eds. Lecture Notes in Artificial Intelligence, vol. 4483. Springer-Verlag, 266–271.
- GELFOND, M. AND LEONE, N. 2002. Logic programming and knowledge representation – the A-Prolog perspective. *Artificial Intelligence* 138, 1-2, 3–38.
- GELFOND, M. AND LIFSCHITZ, V. 1988. The stable model semantics for logic programming. In *Logic Programming: Proceedings of the Fifth International Conf. and Symp.*, R. Kowalski and K. Bowen, Eds. 1070–1080.
- GIUNCHIGLIA, E., LIERLER, Y., AND MARATEA, M. 2004. Sat-based answer set programming. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*. AAAI Press / The MIT Press, 61–66.
- GIUNCHIGLIA, E. AND MARATEA, M. 2005. On the Relation Between Answer Set and SAT Procedures (or, Between Cmodels and smodels). In *Logic Programming, 21st International Conference, ICLP 2005, Sitges, Spain, October 2-5, 2005, Proceedings*, M. Gabrielli and G. Gupta, Eds. Lecture Notes in Computer Science, vol. 3668. Springer, 37–51.
- GRAS, D. C. AND HERMENEGILDO, M. V. 2000. A new module system for prolog. In *Computational Logic - CL 2000, First International Conference, London, UK, 24-28 July, 2000, Proceedings*, J. W. Lloyd, V. Dahl, U. Furbach, M. Kerber, K.-K. Lau, C. Palamidessi, L. M. Pereira, Y. Sagiv, and P. J. Stuckey, Eds. Lecture Notes in Computer Science, vol. 1861. Springer, 131–148.
- GUO, H., RAMAKRISHNAN, C., AND RAMAKRISHNAN, I. 2001. Speculative beats Conservative Justification. In *International Conference on Logic Programming*. Springer Verlag, 150–165.
- HELJANKO, K. AND NIEMELÄ, I. 2003. Bounded LTL model checking with stable models. *Theory and Practice of Logic Programming* 3, 4,5, 519–550.
- KONCZAK, K., LINKE, T., AND SCHAUB, T. 2006. Graphs and Colorings for Answer Set Programming. *Theory and Practice of Logic Programming* 6, 1-2, 61–106.
- LIFSCHITZ, V. 1999. Answer set planning. In *International Conference on Logic Programming*. 23–37.
- LIFSCHITZ, V. 2002. Answer set programming and plan generation. *Artificial Intelligence* 138, 1-2, 39–54.

- LIN, F. AND ZHAO, Y. 2002. ASSAT: Computing Answer Sets of A Logic Program By SAT Solvers. In *AAAI*. 112–117.
- LLOYD, J. 1987. *Foundations of logic programming*. Springer Verlag. Second, extended edition.
- MALLET, S. AND DUCASSÉ, M. 1999. Generating deductive database explanations. In *International Conference on Logic Programming*. 154–168.
- MAREK, V. AND TRUSZCZYŃSKI, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-year Perspective*. 375–398.
- MERA, E., LOPEZ-GARCIA, P., PUEBLA, G., CARRO, M., AND HERMENEGILDO, M. 2006. Using Combined Static Analysis and Profiling for Logic Program Execution Time Estimation. In *International Conference on Logic Programming*. Springer Verlag, 431–432.
- NIEMELÄ, I. 1999. Logic programming with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence* 25, 3,4, 241–273.
- NIEMELÄ, I. AND SIMONS, P. 1997. SMOBELS - an implementation of the stable model and well-founded semantics for normal logic programs. In *Proceedings of ICLP & LPNMR*. 420–429.
- PEMMASANI, G., GUO, H.-F., DONG, Y., RAMAKRISHNAN, C. R., AND RAMAKRISHNAN, I. V. 2004. Online justification for tabled logic programs. In *Functional and Logic Programming, 7th International Symposium, FLOPS 2004, Nara, Japan, April 7-9, 2004, Proceedings*, Y. Kameyama and P. J. Stuckey, Eds. Lecture Notes in Computer Science, vol. 2998. Springer, 24–38.
- PERRI, S., RICCA, F., TERRACINA, G., CIANNI, D., AND VELTRI, P. 2007. An integrated graphic tool for developing and testing DLV programs. In *Proceedings of the 1st SEA Workshop, LPNMR'07*. 86–100.
- PINEDA, A. 1999. Object-oriented programming library O'Ciao. Tech. Rep. CLIP 6/99.0, UPM Madrid.
- PUEBLA, G., BUENO, F., AND HERMENEGILDO, M. V. 1998. A framework for assertion-based debugging in constraint logic programming. In *Principles and Practice of Constraint Programming - CP98, 4th International Conference, Pisa, Italy, October 26-30, 1998, Proceedings*, M. J. Maher and J.-F. Puget, Eds. Lecture Notes in Computer Science, vol. 1520. Springer, 472.
- ROYCHOUDHURY, A., RAMAKRISHNAN, C. R., AND RAMAKRISHNAN, I. V. 2000. Justifying proofs using memo tables. In *PPDP*. 178–189.
- SAHA, D. AND RAMAKRISHNAN, C. R. 2005. Symbolic support graph: A space efficient data structure for incremental tabled evaluation. In *Logic Programming, 21st International Conference, ICLP 2005, Sitges, Spain, October 2-5, 2005, Proceedings*, M. Gabrielli and G. Gupta, Eds. Lecture Notes in Computer Science, vol. 3668. Springer, 235–249.
- SHAPIRO, E. Y. 1982. Algorithmic program diagnosis. In *POPL*. 299–308.
- SIMONS, P., NIEMELÄ, N., AND SOININEN, T. 2002. Extending and Implementing the Stable Model Semantics. *Artificial Intelligence* 138, 1–2, 181–234.
- SPECHT, G. 1993. Generating explanation trees even for negations in deductive database systems. In *LPE*. 8–13.
- SYRJÄNEN, T. 2006. Debugging inconsistent answer set programs. In *Proceedings of the 11th International Workshop on Non-Monotonic Reasoning*. Lake District, UK, 77–84.
- VAN EMDEN, M. AND KOWALSKI, R. 1976. The semantics of predicate logic as a programming language. *Journal of the ACM*. 23, 4, 733–742.
- VAN GELDER, A., ROSS, K., AND SCHLIPF, J. 1991. The well-founded semantics for general logic programs. *Journal of ACM* 38, 3, 620–650.
- VAUPEL, R., PONTELLI, E., AND GUPTA, G. 1997. Visualization of and/or-parallel execution of logic programs. In *ICLP*. 271–285.

Appendix: Proofs

Proof of Proposition 1.

Proposition 1. Given a program P and an answer set M of P , the well-founded model of $NR(P, \mathcal{TA}_P(M))$ is equal to M .

The following result has been proved (Apt and Bol 1994).

Theorem 1

Let P be a program and j be the first index such that $(K_j, U_j) = (K_{j+1}, U_{j+1})$. The well-founded model of P , $WF_P = \langle W^+, W^- \rangle$, satisfies $W^+ = K_j$ and $W^- = \mathcal{A} \setminus U_j$.

Let T_R denote the traditional immediate consequence operator for a definite program R (Lloyd 1987). We will also make use of the usual notations such as $T_R \uparrow 0 = \emptyset$, $T_R \uparrow i = T_R(T_R \uparrow (i-1))$. Given a program P and one of its answer sets M , to simplify the presentation, let us denote with $Q(M)$ the negative reduct $NR(P, \mathcal{TA}_P(M))$. We will denote with (K_i^P, U_i^P) the pair (K_i, U_i) (Definition 2) for the original program P and with (K_i^Q, U_i^Q) the pair (K_i, U_i) for program $Q(M)$ respectively.

Lemma 8.1

For a program P , $lfp(T_{P^+}) = lfp(T_{Q(M)^+})$.

Proof. Clearly, $lfp(T_{P^+}) \supseteq lfp(T_{Q(M)^+})$ since $P^+ \supseteq Q(M)^+$.

Let us prove, by induction on i , that $T_{P^+} \uparrow i \subseteq T_{Q(M)^+} \uparrow i$. The result is trivial for the base case. Let us assume that the result holds for i and let us prove it for $i+1$. Consider $a \in T_{P^+} \uparrow i+1$. This means that there is a rule $r \in P^+$ such that $head(r) = a$ and $pos(r) \subseteq T_{P^+} \uparrow i$. Since $a \in lfp(T_{P^+}) \subseteq M^+$, we know that $a \in M^+$ and therefore $r \in Q(M)^+$. Thus, thanks to the inductive hypothesis, we can conclude that $a \in T_{Q(M)^+} \uparrow i+1$. \square

Corollary 1

For a program P , $K_0^P = K_0^Q$.

Lemma 8.2

For a program P , $U_0^Q \subseteq U_0^P$.

Proof. We prove, by induction on i , that $T_{Q(M), K_0^Q} \uparrow i \subseteq T_{P, K_0^P} \uparrow i$.

Base: The result is obvious for $i=0$, since

$$T_{Q(M), K_0^Q} \uparrow 0 = \emptyset = T_{P, K_0^P} \uparrow 0$$

Let $a \in T_{Q(M), K_0^Q} \uparrow 1$. This implies that there is $r \in Q(M)$ such that $head(r) = a$, $pos(r) = \emptyset$, and $neg(r) \cap K_0^Q = \emptyset$. Since $Q(M) \subseteq P$, we also have that $r \in P$. Furthermore, since $K_0^P = K_0^Q$ (from Corollary 1), we have that $a \in T_{P, K_0^P} \uparrow 1$.

Step: Let us assume the result to be true for i and let us consider the iteration $i+1$. Let $a \in T_{Q(M), K_0^Q} \uparrow i+1$. This implies that there is a rule $r \in Q(M)$ such that

- $head(r) = a$,

- $pos(r) \subseteq T_{Q(M), K_0^Q} \uparrow i$, and
- $neg(r) \cap K_0^Q = \emptyset$.

Since $Q(M) \subseteq P$, then we have that $r \in P$. Furthermore, since $K_0^P = K_0^Q$, we have that $a \in T_{P, K_0^P} \uparrow i + 1$. \square

Proposition 5

For every i , $U_i^Q \subseteq U_i^P$ and $K_i^P \subseteq K_i^Q$.

Proof. We will prove this result by induction on i . The base case follows directly from Lemmas 8.1-8.2. Let us proceed with the inductive step.

First, let us prove by induction on j that $T_{P, U_{i-1}^P} \uparrow j \subseteq T_{Q(M), U_{i-1}^Q} \uparrow j$.

- **Base:** Let $a \in T_{P, U_{i-1}^P} \uparrow 1$. This implies that there is a rule $r \in P$ such that $head(r) = a$, $pos(r) = \emptyset$, and $neg(r) \cap U_{i-1}^P = \emptyset$. Since $K_i^P \subseteq W^+$, we have that $a \notin \mathcal{TA}(M)$, and thus $r \in Q(M)$. Furthermore, since $U_{i-1}^Q \subseteq U_{i-1}^P$, we have that $neg(r) \cap U_{i-1}^Q = \emptyset$. Hence, $a \in T_{Q(M), U_{i-1}^Q} \uparrow 1$.
- **Step:** let us assume the result to hold for j and let us prove it for $j + 1$. Let $a \in T_{P, U_{i-1}^P} \uparrow j + 1$. This implies that there is a rule $r \in P$ such that
 - $head(r) = a$,
 - $pos(r) \subseteq T_{P, U_{i-1}^P} \uparrow j$, and
 - $neg(r) \cap U_{i-1}^P = \emptyset$.

Since $K_i^P \subseteq W^+$, we have that $a \notin \mathcal{TA}(M)$, and thus $r \in Q(M)$. Furthermore, since $U_{i-1}^Q \subseteq U_{i-1}^P$, we have that $neg(r) \cap U_{i-1}^Q = \emptyset$. By inductive hypothesis, we have that $pos(r) \subseteq T_{Q(M), U_{i-1}^Q} \uparrow j$. Hence, $a \in T_{Q(M), U_{i-1}^Q} \uparrow j + 1$.

Let us now prove, by induction on j that $T_{Q(M), K_i^Q} \uparrow j \subseteq T_{P, K_i^P} \uparrow j$.

- **Base:** Let $a \in T_{Q(M), K_i^Q} \uparrow 1$. This implies that there is a rule $r \in Q(M)$ such that $head(r) = a$, $pos(r) = \emptyset$, and $neg(r) \cap K_i^Q = \emptyset$. Since $Q(M) \subseteq P$, we have that $r \in P$. Furthermore, since $K_i^P \subseteq K_i^Q$, we have that $neg(r) \cap K_i^P = \emptyset$. Hence, $a \in T_{P, K_i^P} \uparrow 1$.
- **Step:** let us assume the result to hold for j and let us consider the case $j + 1$. Let $a \in T_{Q(M), K_i^Q} \uparrow j + 1$. This implies that there is a rule $r \in Q(M)$ such that $head(r) = a$, $pos(r) \subseteq T_{Q(M), K_i^Q} \uparrow j$, and $neg(r) \cap K_i^Q = \emptyset$. Since $Q(M) \subseteq P$, we have that $r \in P$. Furthermore, since $K_i^P \subseteq K_i^Q$, we have that $neg(r) \cap K_i^P = \emptyset$. By inductive hypothesis, we also have that $pos(r) \subseteq T_{P, K_i^P} \uparrow j$. Hence, $a \in T_{P, K_i^P} \uparrow j + 1$.

\square

Lemma 8.3

If M is an answer set of P , then M is an answer set of $Q(M)$.

Proof. Obviously, $lfp(T_{Q(M)^{M^+}}) \subseteq lfp(T_{P^{M^+}})$ because $Q(M)^{M^+} \subseteq P^{M^+}$. Thus, it is sufficient to show that $lfp(T_{P^{M^+}}) \subseteq lfp(T_{Q(M)^{M^+}})$. We prove by induction on i that $T_{P^{M^+}} \uparrow i \subseteq T_{Q(M)^{M^+}} \uparrow i$.

Base: Let $a \in T_{P^{M^+}} \uparrow 1$. This implies that there is a rule $r \in P$ such that $head(r) = a$, $pos(r) = \emptyset$, and $neg(r) \subseteq M^-$. Because $a \in M^+$, we have that $r \in Q(M)$. Thus, $a \in T_{Q(M)^{M^+}} \uparrow 1$.

Step: Let $a \in T_{P^{M^+}} \uparrow i + 1$. This implies that there exists a rule $r \in P$ such that $head(r) = a$, $pos(r) \subseteq T_{P^{M^+}} \uparrow i$, and $neg(r) \subseteq M^-$. Since $a \in M^+$, we have that $r \in Q(M)$. Thus, $a \in T_{Q(M)^{M^+}} \uparrow i + 1$. \square

Let us indicate in the rest of this discussion the well-founded model of $Q(M)$ with WF_Q and the well-founded model of P with WF_P .

Lemma 8.4

$$\mathcal{TA}_P(M) \subseteq WF_Q^-.$$

Proof. Consider $a \in \mathcal{TA}_P(M)$. We have that $a \notin U_i^Q$ for every i since there are no rules with a as head in $Q(M)$. This means that $a \in WF_Q^-$. Thus, $\mathcal{TA}_P(M) \subseteq WF_Q^-$. \square

Proposition 6

The well-founded model WF_Q of $Q(M)$ is equal to M , i.e., $W_Q = M$.

Proof. From Proposition 5, we have that $WF_P^+ \subseteq WF_Q^+$ and $WF_P^- \subseteq WF_Q^-$. Furthermore, since $\mathcal{TA}_P(M) \subseteq WF_Q^-$, we can conclude that $M^- \subseteq WF_Q^-$. Since M is an answer set of $Q(M)$, we also have that $WF_Q^- \subseteq M^-$. Thus, $M^- = WF_Q^-$. This conclusion implies that there is a value k such that $U_k^Q = \mathcal{A} \setminus M^-$.

Let us now show that $K_{k+1}^Q = M^+$. Since M is an answer set of $Q(M)$, we immediately have that $K_{k+1}^Q \subseteq M^+$. Let us prove, by induction on i , that $T_{P^{M^+}} \uparrow i \subseteq T_{Q(M), U_k^Q} \uparrow i$.

Base: Let $a \in T_{P^{M^+}} \uparrow 1$. This implies that there is a rule $r \in P$ such that $head(r) = a$, $pos(r) = \emptyset$, and $neg(r) \subseteq M^-$. Since $a \in M^+$, we have that $r \in Q(M)$. Furthermore, since $U_k^Q = \mathcal{A} \setminus M^-$ and $neg(r) \subseteq M^-$, we have that $neg(r) \cap U_k^Q = \emptyset$. Thus, $a \in T_{Q(M), U_k^Q} \uparrow 1$.

Step: Let $a \in T_{P^{M^+}} \uparrow i + 1$. This implies that there is a rule $r \in P$ such that $head(r) = a$, $pos(r) \subseteq T_{P^{M^+}} \uparrow i$, and $neg(r) \subseteq M^-$. Since $a \in M^+$, we have that $r \in Q(M)$. Furthermore, since $U_k^Q = \mathcal{A} \setminus M^-$ and $neg(r) \subseteq M^-$, we have that $neg(r) \cap U_k^Q = \emptyset$. By inductive hypothesis, we also have that $pos(r) \subseteq T_{Q(M), U_k^Q} \uparrow i$. Thus, $a \in T_{Q(M), U_k^Q} \uparrow i + 1$. \square

Proof of Lemma 4.1.

The proof of this lemma makes use of several results and definitions in (Brass et al. 2001). For this reason, let us recall the necessary definitions from (Brass et al. 2001). Given a program P , let us denote with $heads(P) = \{a \mid \exists r \in P. head(r) = a\}$ and with $facts(P) = \{a \mid (a :-) \in P\}$. We can consider the following program transformations (Brass et al. 2001):

- $P_1 \mapsto_P P_2$ iff $a :- body \in P_1$, not $b \in body$, $b \notin heads(P_1)$, and $P_2 = (P_1 \setminus \{a :- body\}) \cup \{a :- body \setminus \{not b\}\}$
- $P_1 \mapsto_N P_2$ iff $a :- body \in P_1$, not $b \in body$, $b \in facts(P_1)$, and $P_2 = P_1 \setminus \{a :- body\}$
- $P_1 \mapsto_S P_2$ iff $a :- body \in P_1$, $b \in body$, $b \in facts(P_1)$, and $P_2 = (P_1 \setminus \{a :- body\}) \cup \{a :- (body \setminus \{b\})\}$
- $P_1 \mapsto_F P_2$ iff $a :- body \in P_1$, $b \in body$, $b \notin heads(P_1)$, and $P_2 = P_1 \setminus \{a :- body\}$
- $P_1 \mapsto_L P_2$ iff there is a non-empty set of atoms S such that
 - for each rule $a :- body$ in P_1 where $a \in S$ we have that $S \cap body \neq \emptyset$
 - $P_2 = \{r \in P_1 \mid body(r) \cap S = \emptyset\}$
 - $P_1 \neq P_2$

We write $P_1 \mapsto P_2$ to indicate that there exists a transformation $t \in \{P, N, S, F, L\}$ such that $P_1 \mapsto_t P_2$. A program P is *irreducible* if $P \mapsto_t P$ for every $t \in \{P, N, S, F, L\}$. The results in (Brass et al. 2001) show that the above transformation system is terminating and confluent, i.e., given a program P , (a) there exists a sequence of programs $P = P_0, P_1, \dots, P_n = P^*$ such that $P_i \mapsto P_{i+1}$ for $0 \leq i \leq n-1$ and P^* is irreducible; and (b) for every sequence of programs $P = Q_0, Q_1, \dots, Q_m = Q^*$ such that $Q_i \mapsto Q_{i+1}$ for $0 \leq i \leq m-1$ and Q^* is irreducible then $P^* = Q^*$. We call the irreducible program P^* obtained from P through this transformation system the normal form of P . The result in (Brass et al. 2001) shows that the well-founded model $WF_P = \langle W^+, W^- \rangle$ of P can be obtained by

$$W^+ = facts(P^*) \quad W^- = \{a \mid a \notin heads(P^*)\}$$

where P^* is the normal form of P .

Lemma 4.1. Let P be a program, M an answer set, and WF_P the well-founded model of P . Each atom $a \in WF_P$ has an off-line justification w.r.t. M and \emptyset which does not contain any negative cycle.

Proof: Let us consider the sequence of transformations of the program

$$P = P_0 \mapsto P_1 \mapsto \dots \mapsto P^*$$

such that the transformation \mapsto_L is used only when no other transformation can be applied. Furthermore, let

$$WP_i = \langle W_i^+, W_i^- \rangle = \langle facts(P_i), \{a \mid a \notin heads(P_i)\} \rangle$$

We wish to prove, by induction on i , that if $a \in W_i^+ \cup W_i^-$ then it has a justification which is free of negative cycles and it contains exclusively elements in $W_i^+ \cup W_i^-$.

For the sake of simplicity, we will describe justification graphs simply as set of edges. Also, we will denote with $\mathcal{J}(a)$ the graph created for the element a .

Base: Let us consider $i = 0$. We have two cases:

- $a \in W_0^+$. This means that $a \in facts(P_0) = facts(P)$. This implies that $\mathcal{J}(a) = \{(a^+, \top, +)\}$ is a cycle-free justification for a w.r.t. WP_0 and \emptyset .
- $a \in W_0^-$. This means that $a \notin heads(P_0) = heads(P)$. From the definition of off-line justification, this means that we can build the justification $\mathcal{J}(a) = \{(a^-, \perp, +)\}$, which is also cycle-free. In addition, the only atoms in the justification belongs to $W_0^+ \cup W_0^-$.

Step: Let us assume that the inductive hypothesis holds for $j \leq i$. Let us consider $a \in W_{i+1}^+ \cup W_{i+1}^-$. We have two cases:

- $a \in W_{i+1}^+$. Without loss of generality, we can assume that $a \notin W_i^+$. This means that the reduction step taken to construct P_{i+1} from P_i produced a fact of the form $a :-$. This implies that there exists a rule

$$a :- b_1, \dots, b_k, not\ c_1, \dots, not\ c_h$$

in P such that each b_j has been removed in the previous steps by \mapsto_S transformations, and each $not\ c_r$ has been removed by \mapsto_P transformations. This means that each $b_j \in W_i^+$, each $c_r \in W_i^-$, and, by inductive hypothesis, they admit justifications free of negative cycles. We can construct a justification $\mathcal{J}(a)$ for a , which is free of negative cycles and is the union of all the justifications free of negative cycles of $b_1, \dots, b_k, c_1, \dots, c_h$ and the edges $(a^+, b_1^+, +), \dots, (a^+, b_k^+, +), (a^+, c_1^-, -), \dots, (a^+, c_h^-, -)$. Note that, with the exception of a , the atoms involved in the justification $\mathcal{J}(a)$ are only atoms of $W_i^+ \cup W_i^-$.

- Let us now consider $a \in W_{i+1}^-$. Again, we assume that $a \notin W_i^-$. This means that in P_{i+1} there are no rules left for a . Let us consider each individual rule for a in P , of the generic form

$$a :- b_1, \dots, b_k, not\ c_1, \dots, not\ c_h \quad (2)$$

We consider two cases:

- $P_i \mapsto_N P_{i+1}$ or $P_i \mapsto_F P_{i+1}$. By our assumption about the sequence of transformations, we can conclude that the transformation \mapsto_L has not been applied in removing rules whose head is a . In other words, each rule (2) has been removed by either a \mapsto_N or a \mapsto_F transformation. This implies that for each rule (2), there exists either a $c_j \in W_i^+$ or a $b_l \in W_i^-$, i.e., there exists $C^+ \subseteq W_i^+$ and $C^- \subseteq W_i^-$ such that for each rule r with $head(r) = a$, $C^+ \cap neg(r) \neq \emptyset$ or $C^- \cap pos(r) \neq \emptyset$. Without loss of generality, we can assume that C^+ and C^- are minimal (w.r.t. \subseteq). By inductive hypothesis, we know that each element in C^+ and C^- posses a justification free of negative cycles which contain only atoms in WP_i . Similar to the first item, we have that $\mathcal{J}(a) = \bigcup_{c \in C^+ \cup C^-} \mathcal{J}(c) \cup \{(a^-, c^+, -) \mid c \in C^+\} \cup \{(a^-, c^-, +) \mid c \in C^-\}$ is a justification free

of negative cycles for a which, with the exception of a^- , contains only atoms in WP_i .

- $P_i \mapsto_L P_{i+1}$. The fact that $a \in W_{i+1}^- \setminus W_i^-$ indicates that all rules with a as head have been removed. In this case, there might be some rules with a as its head that have been removed by other transformations. Let $R_1(a)$ (resp. $R_2(a)$) be the set of rules, whose head is a , which are removed by a transformation \mapsto_F or \mapsto_N (resp. \mapsto_L).

Let S be the set of atoms employed for the \mapsto_L step (i.e., the i -th step). Let a_1, \dots, a_s be an enumeration of S . For a subset X of S , let $\min(X)$ denote the element in X with the smallest index according to the above enumeration.

Let

$$\begin{aligned} G_0 &= \{(a^-, b^-, +) \mid a :- \text{body} \in P_i, b = \min(\text{body} \cap S)\} \\ G_{j+1} &= \{(b^-, c^-, +) \mid \exists (d^-, b^-, +) \in G_j, (b :- \text{body}) \in P_i, \\ &\quad c = \min(\text{body} \cap S)\} \end{aligned}$$

Because of the finiteness of S , there exists some j such that $G_j \subseteq \bigcup_{0 \leq i \leq j-1} G_i$. Let be the graph⁵ $G = \bigcup_{j \geq 0} G_j$. Because of the property of S , it is easy to see that for each atom c in the graph G , $\text{support}(c, G)$ is a LCE of c w.r.t. WP_i and \emptyset (w.r.t. the program P_i). Thus, we have that G is an off-line justification for a in P_i . Furthermore, it contains only positive cycles and it is composed of atoms from $S \cup \{a\}$.

The construction of G takes care of rules of the form (2), which belong to $R_2(a)$. Similar to the previous case, we know that for each atom b such that b^- is a node in G , there exists $C_b^+ \subseteq W_{i-1}^+$ and $C_b^- \subseteq W_{i-1}^-$ such that for each rule r with $\text{head}(r) = b$ in $R_1(b)$, $C_b^+ \cap \text{neg}(r) \neq \emptyset$ or $C_b^- \cap \text{pos}(r) \neq \emptyset$. G can be extended to an off-line justification of a by adding to its the justifications of other atoms that falsify the rules in $R_1(b)$ for every $b \in S$. More precisely, for each atom b such that b^- is a node in G , let

$$G_b = \bigcup_{c \in C_b^+ \cup C_b^-} \mathcal{J}(c) \cup \{(b^-, c^+, -) \mid c \in C_b^+\} \cup \{(b^-, c^-, +) \mid c \in C_b^-\}.$$

Note that $\mathcal{J}(c)$ in the above equation exists due to the inductive hypothesis. Furthermore, each G_b contains only atoms in WP_i with the exception of b and therefore cannot contain negative cycles. Thus, $G' = G \cup \bigcup_{b \text{ is a node in } G} G_b$ does not contain negative cycles. It is easy to check that $\text{support}(c, G')$ is a LCE of c in P w.r.t. WP_{i+1} and \emptyset . Thus, G' is an off-line justification for a in P w.r.t. WP_{i+1} and \emptyset .

□

⁵ Again, we define the graph by its set of edges.

Proof of Proposition 2.

Proposition 2. Let P be a program and M an answer set. For each atom a , there is an off-line justification w.r.t. M and $\mathcal{TA}_P(M)$ which does not contain negative cycles.

Proof: The result is trivial, since all the elements in $\mathcal{TA}_P(M)$ are immediately set to false, and $NR(P, \mathcal{TA}_P(M))$ has a well-founded model equal to M (and thus all elements have justifications free of negative cycles, from Lemma 4.1). \square

Proof of Proposition 3.

The proof of this proposition will develop through a number of intermediate steps. Let us start by introducing some notation. Given a program P and given the Herbrand universe \mathcal{A} , let $nohead(P) = \{a \in \mathcal{A} : \forall r \in P. a \neq head(r)\}$. Furthermore, for two sets of atoms Γ, Δ such that $\Gamma \cap \Delta = \emptyset$, we define a program transformation $\rightarrow_{\langle \Gamma, \Delta \rangle}$ as follows. The program P' , obtained from P by

- removing r from P if $pos(r) \cap \Delta \neq \emptyset$ or $neg(r) \cap \Gamma \neq \emptyset$ (remove rules that are inapplicable w.r.t. $\langle \Gamma, \Delta \rangle$).
- replacing each remaining rule r with r' where $head(r') = head(r)$, $pos(r') = pos(r) \setminus \Gamma$, and $neg(r') = neg(r) \setminus \Delta$ (normalize the body of the rules w.r.t. $\langle \Gamma, \Delta \rangle$)

is said to be the result of the transformation $\rightarrow_{\langle \Gamma, \Delta \rangle}$. We write $P \rightarrow_{\langle \Gamma, \Delta \rangle} P'$ to denote this fact.

The following can be proven.

Lemma 8.5

Let P be a program Γ and Δ be two sets of atoms such that $\Gamma \subseteq facts(P)$, $\Delta = \bigcup_{i=1}^k S_i \cup X$ where $X \subseteq nohead(P)$ and S_1, \dots, S_k is a sequence of sets of atoms such that $S_i \in cycles(\langle \emptyset, \emptyset \rangle)$ for $1 \leq i \leq k$. It holds that if $P \rightarrow_{\langle \Gamma, \Delta \rangle} P'$ then there exists a sequence of basic transformations $P \mapsto_{t_1} P_1 \mapsto_{t_2} \dots \mapsto_{t_m} P'$ where $t_i \in \{P, N, S, F, L\}$ (see the proof of Lemma 4.1 for the definition of these transformations).

Proof. We prove this lemma by describing the sequence of transformations \mapsto . Let $\Omega = \bigcup_{i=1}^k S_i$. The proof is based on the following observations:

1. Since Γ is a set of facts, we can repeatedly apply the \mapsto_N and \mapsto_S transformations to P . The result is a program P_1 with the following properties: for every $r \in P_1$, there exists some $r' \in P$ with $neg(r') \cap \Gamma = \emptyset$
 - (a) $neg(r) = neg(r')$
 - (b) $head(r) = head(r')$ and
 - (c) $pos(r) = pos(r') \setminus \Gamma$.
2. Since X is a set of atoms with no rules in P_1 , we can repeatedly apply the \mapsto_P and \mapsto_F transformations to P_1 for the atoms belonging to X . The result is a program P_2 with the following properties: for every $r \in P_2$, there exists some $r' \in P_1$ with $pos(r') \cap X = \emptyset$ and
 - (a) $pos(r) = pos(r')$
 - (b) $head(r) = head(r')$ and
 - (c) $neg(r) = neg(r') \setminus X$.
3. Since Ω is a set of atoms with cycles, we can apply the loop detection transformation \mapsto_L for each of the loops in Ω to P_2 ; thus, we obtain $P_3 = P_2 \setminus \{r \in P_2 \mid head(r) \in \Omega\}$.

4. Since atoms in Ω will no longer have defining rules in P_3 , the transformations for atoms in Ω (similar to those for atoms in X) can be applied to P_3 ; the result is the program P_4 with the property: for every $r \in P_4$, there exists some $r' \in P_3$ with $pos(r') \cap \Omega = \emptyset$ and

- (a) $pos(r) = pos(r')$
- (b) $head(r) = head(r')$ and
- (c) $neg(r) = neg(r') \setminus \Omega$.

Finally, let us consider P_4 ; for each rule $r \in P_4$, there is a rule $r' \in P$ such that $pos(r') \cap \Delta = \emptyset$, $neg(r') \cap \Gamma = \emptyset$, and

- 1. $pos(r) = pos(r') \setminus \Gamma$
- 2. $head(r) = head(r')$ and
- 3. $neg(r) = neg(r') \setminus \Delta$.

This shows that $P \rightarrow_{\langle \Gamma, \Delta \rangle} P_4$. □

For a program P , let WF_P be its well-founded model. Let us define a sequence of programs $P_0, P_1, \dots, P_k, \dots$ as follows:

$$\begin{array}{rcl} P_0 & = & P \\ P_0 & \rightarrow_{\langle \Gamma^1(WF_P), \Delta^1(WF_P) \rangle} & P_1 \\ P_i & \rightarrow_{\langle \Gamma^{i+1}(WF_P), \Delta^{i+1}(WF_P) \rangle} & P_{i+1} \end{array}$$

Lemma 8.6

Given the previously defined sequence of programs, the following properties hold:

- 1. For $i \geq 0$, $\Gamma^i(WF_P) \subseteq facts(P_i)$ and $\Delta^i(WF_P) \subseteq nohead(P_i)$.
- 2. If $\Gamma^i(WF_P) = \Gamma^{i+1}(WF_P)$ then $\Gamma^{i+1}(WF_P) = facts(P_{i+1})$.
- 3. If $\Delta^i(WF_P) = \Delta^{i+1}(WF_P)$ then $\Delta^{i+1}(WF_P) = nohead(P_{i+1})$.

Proof.

- 1. The first property holds because of the construction of P_i and the definitions of $\Gamma^i(WF_P)$ and $\Delta^i(WF_P)$.
- 2. Consider some $a \in facts(P_{i+1})$. By the definition of P_{i+1} , there exists some rule $r \in P_i$ such that
 - $head(r) = a$,
 - $pos(r) \cap \Delta^i(WF_P) = \emptyset$,
 - $neg(r) \cap \Gamma^i(WF_P) = \emptyset$,
 - $pos(r) \setminus \Gamma^i(WF_P) = \emptyset$, and
 - $neg(r) \setminus \Delta^i(WF_P) = \emptyset$.

This implies that $pos(r) \subseteq \Gamma^i(WF_P)$ and $neg(r) \subseteq \Delta^i(WF_P)$, i.e., $a \in \Gamma^{i+1}(WF_P)$. This proves the equality of the second item.

- 3. Consider some $a \in nohead(P_{i+1})$. This means that every rule of P_i having a in the head has been removed; i.e., for every $r \in P_i$ with $head(r) = a$, we have that

- $pos(r) \cap \Delta^i(WF_P) \neq \emptyset$ or
- $neg(r) \cap \Gamma^i(WF_P) \neq \emptyset$.

This implies that $a \in \Delta^{i+1}(WF_P)$, which allows us to conclude the third property. □

Lemma 8.7

Let k be the first index such that $\Gamma^k(WF_P) = \Gamma^{k+1}(WF_P)$ and $\Delta^k(WF_P) = \Delta^{k+1}(WF_P)$. Then, P_{k+1} is irreducible w.r.t. the transformations \mapsto_{NPSFL} .

Proof. This results follows from Lemma 8.6, since $\Gamma^{k+1}(WF_P) = facts(P_{k+1})$ and $\Delta^{k+1}(WF_P) = nohead(P_{k+1})$. This means that $P_{k+1} \mapsto_{NPSF}^* P_{k+1}$. Furthermore, $cycles(\langle \Gamma^{k+1}(WF_P), \Delta^{k+1}(WF_P) \rangle) = \emptyset$. Hence, P_{k+1} is irreducible. □

Lemma 8.8

For a program P , $WF_P = \langle \Gamma(WF_P), \Delta(WF_P) \rangle$.

Proof. This results follows from Lemmas 8.6 and 8.7. □

Lemma 8.9

Given two p-interpretations $I \sqsubseteq J$, we have that $\Gamma(I) \subseteq \Gamma(J)$ and $\Delta(I) \subseteq \Delta(J)$.

Proof. We prove that $\Gamma^i(I) \subseteq \Gamma^i(J)$ and $\Delta^i(I) \subseteq \Delta^i(J)$ by induction on i .

1. **Base:** $i = 0$. This step is obvious, since $I \subseteq J$.
2. **Step:** Let $I_i = \langle \Gamma^i(I), \Delta^i(I) \rangle$ and $J_i = \langle \Gamma^i(J), \Delta^i(J) \rangle$. From the inductive hypothesis, we can conclude that $I_i \sqsubseteq J_i$. This result, together with the fact that, for any rule r , $I_i \models body(r)$ implies $J_i \models body(r)$, allows us to conclude that $\Gamma^{i+1}(I) \subseteq \Gamma^{i+1}(J)$. Similarly, from the fact that $cycles(I_i) \subseteq cycles(J_i)$ and the inductive hypothesis, we can show that $\Delta^{i+1}(I) \subseteq \Delta^{i+1}(J)$. □

Lemma 8.10

Given a program P and an answer set M of P , $M = \langle \Gamma(M), \Delta(M) \rangle$.

Proof. Let us prove this lemma by contradiction. Let $J = \langle \Gamma(M), \Delta(M) \rangle$. First, Lemma 8.9 and 8.8 imply that $WF_P \sqsubseteq J$. Since M is an answer set of P , there exists some level mapping ℓ such that M is a *well-supported model* w.r.t. ℓ (Fages 1994), i.e., for each $a \in M^+$ there exists a rule r_a satisfying the following conditions:

- $head(r_a) = a$,
- r_a is supported by M (i.e., $pos(r_a) \subseteq M^+$ and $neg(r_a) \subseteq M^-$), and
- $\ell(a) > \ell(b)$ for each $b \in pos(r_a)$.

We have to consider the following cases:

- **Case 1:** $M^+ \setminus J^+ \neq \emptyset$. Consider $a \in M^+ \setminus J^+$ such that $\ell(a) = \min\{\ell(b) \mid b \in M^+ \setminus J^+\}$. There exists a rule r such that $\text{head}(r) = a$, r is supported by M , and $\ell(a) > \ell(b)$ for each $b \in \text{pos}(r)$. The minimality of $\ell(a)$ implies that $\text{pos}(r) \subseteq J^+$. The fact that $a \notin J^+$ implies that $\text{neg}(r) \setminus J^- \neq \emptyset$. Consider some $c \in \text{neg}(r) \setminus J^-$. Clearly, $c \notin (NANT(P) \setminus WF_P^-)$ —otherwise, it would belong to J^- . This implies that $c \in WF_P^-$ because $c \in NANT(P)$. Hence, $c \in J^-$. This represents a contradiction.
- **Case 2:** $M^- \setminus J^- \neq \emptyset$. Consider $a \in M^- \setminus J^-$. This is possible only if there exists some rule r such that
 - $\text{head}(r) = a$,
 - $\text{pos}(r) \cap \Delta(M) = \emptyset$,
 - $\text{neg}(r) \cap \Gamma(M) = \emptyset$, and
 - either
 - (i) $\text{neg}(r) \setminus \Delta(M) \neq \emptyset$, or
 - (ii) $\text{pos}(r) \setminus \Gamma(M) \neq \emptyset$.

In what follows, by R_a we denote the set of rules in P whose head is a and whose bodies are neither true nor false in J .

If (i) is true, then there exists some $b \in \text{neg}(r) \setminus \Delta(M)$. Since $b \in \text{neg}(r)$, we have that $b \in NANT(P)$. This implies that $b \notin M^-$ or $b \in WF_P^-$. The second case cannot happen since $WF_P \sqsubseteq J$ (Lemma 8.9). So, we must have that $b \notin M^-$. This means that $b \in M^+$ (since M is an answer set, and thus a complete interpretation), and hence, $b \in J^+$ (Case 1). This contradicts the fact that $\text{neg}(r) \cap \Gamma(M) = \emptyset$. Therefore, we conclude that (i) cannot happen. Since (i) is not true, we can conclude that $R_a \neq \emptyset$ and for every $r \in R_a$ and $b \in \text{pos}(r) \setminus \Gamma(M)$, $b \in J^- \setminus M^-$ and $R_b \neq \emptyset$. Let us consider the following sequence:

$$\begin{aligned}
 C_0 &= \{a\} \\
 C_1 &= \bigcup_{r \in R_a} (\text{pos}(r) \setminus \Gamma(M)) \\
 &\dots \\
 C_i &= \bigcup_{b \in C_{i-1}} (\bigcup_{r \in R_b} (\text{pos}(r) \setminus \Gamma(M)))
 \end{aligned}$$

Let $C = \bigcup_{i=0}^{\infty} C_i$. It is easy to see that for each $c \in C$, it holds that $c \in M^- \setminus J^-$, $R_c \neq \emptyset$, and for each $r \in R_c$, $\text{pos}(r) \cap C \neq \emptyset$. This means that $C \in \text{cycles}(J)$. This is a contradiction with $C \subseteq (M^- \setminus J^-)$. \square

Proposition 3. For a program P , we have that:

- Γ and Δ maintains the consistency of J , i.e., if J is an interpretation, then $\langle \Gamma(J), \Delta(J) \rangle$ is also an interpretation;
- Γ and Δ are monotone w.r.t the argument J , i.e., if $J \sqsubseteq J'$ then $\Gamma(J) \subseteq \Gamma(J')$ and $\Delta(J) \subseteq \Delta(J')$;
- $\Gamma(WF_P) = WF_P^+$ and $\Delta(WF_P) = WF_P^-$; and
- if M is an answer set of P , then $\Gamma(M) = M^+$ and $\Delta(M) = M^-$.

Proof:

1. Follows immediately from the definition of Γ and Δ .
2. Since $J \models \text{body}(r)$ implies $J' \models \text{body}(r)$ and $S \in \text{cycles}(J)$ implies $S \in \text{cycles}(J')$ if $J \subseteq J'$, the conclusion of the item is trivial.
3. This derives from Lemma 8.8.
4. This derives from Lemma 8.10.

□

Proof of Proposition 4.

To prove this proposition, we first prove Lemma 5.1 and 5.2. We need the following definition.

Definition 19 (Subgraph)

Let G be an arbitrary graph whose nodes are in $\mathcal{A}^p \cup \mathcal{A}^n \cup \{assume, \top, \perp\}$ and whose edges are labeled with $+$ and $-$.

Given $e \in \mathcal{A}^+ \cup \mathcal{A}^-$, the *subgraph* of G with root e , denoted by $Sub(e, G)$, is the graph obtained from G by

- (i) removing all the edges of G which do not lie on any path starting from e , and
- (ii) removing all nodes unreachable from e in the resulting graph.

Throughout this section, let I_i denote $\langle \Gamma^i(J), \Delta^i(J) \rangle$. For a set of atoms C and an element $b \in C$, let

$$K(b, C) = \{c \mid c \in C, \exists r \in P. (head(r) = b, c \in pos(r))\}. \quad (3)$$

Lemma 8.11

For a p-interpretation J and $A = \mathcal{TA}_P(J)$, let $\Delta^0(J) = \Delta_1^0 \cup \Delta_2^0 \cup \Delta_3^0$ where

$$\Delta_1^0 = \{a \in \Delta^0(J) \mid PE(a^-, \langle \emptyset, \emptyset \rangle) \neq \emptyset\},$$

$$\Delta_2^0 = \{a \in \Delta^0(J) \mid PE(a^-, \langle \emptyset, \emptyset \rangle) = \emptyset \text{ and } a \in \mathcal{TA}_P(J)\},$$

and

$$\Delta_3^0 = \{a \in \Delta^0(J) \mid PE(a^-, \langle \emptyset, \emptyset \rangle) = \emptyset \text{ and } a \notin \mathcal{TA}_P(J)\}.$$

The following properties hold:

- Δ_1^0 , Δ_2^0 , and Δ_3^0 are pairwise disjoint.
- for each $a \in \Delta_3^0$ there exists a LCE K_a of a^- w.r.t. $\langle \Gamma^0(J), \Delta^0(J) \rangle$ and A such that for each rule $r \in P$ with $head(r) = a$, $pos(r) \cap K_a \neq \emptyset$.

Proof. The first item is trivial thanks to the definition of Δ_1^0 , Δ_2^0 , and Δ_3^0 . For the second item, for $a \in \Delta_3^0$, there exists some $C \in cycles(\langle \emptyset, \emptyset \rangle)$ such that $a \in C$ and $C \subseteq \Delta^0(J)$. From the definition of a cycle, there exists some $K_a \subseteq C \subseteq \Delta^0(J)$ which satisfies the condition of the second item. \square

We will now proceed to prove Lemma 5.1. For each i , we construct a dependency graph Σ_i for elements in $(\Gamma_i(J))^p$ and $(\Delta_i(J))^n$ as follows. Again, we describe a graph by its set of edges. First, the construction of Σ_0 is as follows.

1. for each $a \in \Gamma^0(J)$, Σ_0 contains the edge $(a^+, \top, +)$.
2. let Δ_1^0 , Δ_2^0 , and Δ_3^0 be defined as in Lemma 8.11.
 - (a) For $a \in \Delta_1^0$, Σ_0 contains the edge $(a^-, \perp, -)$;
 - (b) For $a \in \Delta_2^0$, Σ_0 contains the edge $(a^-, assume, -)$.
 - (c) Let $a \in \Delta_3^0$. This implies that there exists some $C \subseteq J^-$ such that $a \in C$ and $C \in cycles(\langle \emptyset, \emptyset \rangle)$. For each $b \in C$, let K_b be an explanation of b^- w.r.t. $\langle \Gamma^0(J), \Delta^0(J) \rangle$ and $\mathcal{TA}_P(J)$ which satisfies the conditions of the second item in Lemma 8.11. Then, Σ_0 contains the set of edges $\bigcup_{b \in C} \{(b^-, c^-, +) \mid c \in K_b\}$.

3. no other edges are added to Σ_0 .

Lemma 8.12

Let J be a p-interpretation and $A = \mathcal{TA}_P(J)$. The following holds for Σ_0 :

1. for each $a \in \Gamma^0(J)$, $Sub(a^+, \Sigma_0)$ is a safe off-line e-graph of a^+ w.r.t. I_0 and A .
2. for each $a \in \Delta^0(J)$, $Sub(a^-, \Sigma_0)$ is a safe off-line e-graph of a^- w.r.t. I_0 and A .

Proof.

- Consider $a \in \Gamma^0(J)$. Since Σ_0 contains $(a^+, \top, +)$ for every $a \in \Gamma^0(J)$ and \top is a sink in Σ_0 , we can conclude that $Sub(a^+, \Sigma_0) = (\{a^+, \top\}, \{(a^+, \top, +)\})$ and $Sub(a^+, \Sigma_0)$ is a safe off-line e-graph of a^+ w.r.t. I_0 and A .
- Consider $a \in \Delta^0(J)$. Let Δ_1^0 , Δ_2^0 , and Δ_3^0 be defined as in Lemma 8.11. There are three cases:
 1. $a \in \Delta_1^0$. Since Σ_0 contains $(a^-, \perp, -)$ and \perp is a sink in Σ_0 , we can conclude that $Sub(a^-, \Sigma_0) = (\{a^-, \perp\}, \{(a^-, \perp, -)\})$ and $Sub(a^-, \Sigma_0)$ is a safe off-line e-graph of a^- w.r.t. I_0 and A .
 2. $a \in \Delta_2^0$. Since Σ_0 contains $(a^-, assume, -)$ and $assume$ is a sink in Σ_0 , we can conclude that $Sub(a^-, \Sigma_0) = (\{a^-, assume\}, \{(a^-, assume, -)\})$ and $Sub(a^-, \Sigma_0)$ is a safe off-line e-graph of a^- w.r.t. I_0 and A .
 3. for $a \in \Delta_3^0$, let $G = Sub(a^-, \Sigma_0) = (N, E)$. It is easy to see that G is indeed a (J, A) -based e-graph of a^- because, from the construction of G , we have that
 - (i) every node in N is reachable from a^- , and
 - (ii) if $b^- \in N$ then $support(b^-, G) = K_b \subseteq N$ is a local consistent explanation of b^- w.r.t. I_0 and A .

The safety of the e-graph derives from the fact that it does not contain any nodes of the form p^+ .

□

To continue our construction, we will need the following lemma.

Lemma 8.13

Let J be a p-interpretation and $A = \mathcal{TA}_P(J)$ and $i > 0$. Let

$$\Delta_1^i = \{a \in \Delta^i(J) \setminus \Delta^{i-1}(J) \mid PE(a^-, I_{i-1}) \neq \emptyset\}$$

and

$$\Delta_2^i = \{a \in \Delta^i(J) \setminus \Delta^{i-1}(J) \mid PE(a^-, I_{i-1}) = \emptyset\}.$$

Then,

- $\Delta_1^i \cap \Delta_2^i = \emptyset$;
- for each $a \in \Delta_1^i$ there exists some LCE K_a of a^- w.r.t. I_i and A such that $\{p \in \mathcal{A} \mid p \in K_a\} \subseteq \Delta^{i-1}(J)$ and $\{a \mid \text{not } a \in K_a\} \subseteq \Gamma^{i-1}(J)$; and

- for each $a \in \Delta_2^i$ there exists some LCE K_a of a^- w.r.t. I_i and A such that $\{p \in \mathcal{A} \mid p \in K_a\} \subseteq \Delta^i(J)$ and $\{a \mid \text{not } a \in K_a\} \subseteq \Gamma^{i-1}(J)$.

Proof. These properties follow immediately from the definition of $\Delta^i(J)$. \square

Given Σ_{i-1} , we can construct Σ_i by reusing all nodes and edges of Σ_{i-1} along with the following nodes and edges.

1. for each $a \in \Gamma^i(J) \setminus \Gamma^{i-1}(J)$, from the definition of $\Gamma^i(J)$ we know that there exists a rule r such that $\text{head}(r) = a$, $\text{pos}(r) \subseteq \Gamma^{i-1}(J)$, and $\text{neg}(r) \subseteq \Delta^{i-1}(J)$. Σ_i contains the node a^+ and the set of edge $\{(a^+, b^+, +) \mid b \in \text{pos}(r)\} \cup \{(a^+, b^-, +) \mid b \in \text{neg}(r)\}$.
2. let Δ_1^i and Δ_2^i be defined as in Lemma 8.13.
 - (a) For $a \in \Delta_1^i$, let K_a be a LCE of a^- satisfying the second condition of Lemma 8.13. Then, Σ_i contains the following set of edges: $\{(a^-, b^-, +) \mid b \in K_a\} \cup \{(a^-, b^+, -) \mid \text{not } b \in K_a\}$;
 - (b) For $a \in \Delta_2^i$, let K_a be a LCE of a^- satisfying the third condition of Lemma 8.13. Then, Σ_i contains the set of links

$$\{(a^-, c^-, +) \mid c \in K_b\} \cup \{(a^-, c^+, -) \mid \text{not } c \in K_b\}.$$

3. no other links are added to Σ_i .

Lemma 8.14

Let J be p-interpretation and $A = \mathcal{TA}_P(J)$. For every integer i , the following properties hold:

1. for each $a \in \Gamma^i(J) \setminus \Gamma^{i-1}(J)$, $\text{Sub}(a^+, \Sigma_i)$ is a safe off-line e-graph of a^+ w.r.t. I_i and A .
2. for each $a \in \Delta^i(J) \setminus \Delta^{i-1}(J)$, $\text{Sub}(a^-, \Sigma_i)$ is a safe off-line e-graph of a^- w.r.t. I_i and A .

Proof. The proof is done by induction on i . The base case is proved in Lemma 8.12. Assume that we have proved the lemma for $j < i$. We now prove the lemma for i . We consider two cases:

1. $a \in \Gamma^i(J) \setminus \Gamma^{i-1}(J)$. Let r be the rule with $\text{head}(r) = a$ used in Item 1 of the construction of Σ_i . For each $b \in \text{pos}(r)$, let $P_b = (NP_b, EP_b) = \text{Sub}(b^+, \Sigma_{i-1})$. For each $b \in \text{neg}(r)$, let $Q_b = (NQ_b, EQ_b) = \text{Sub}(b^-, \Sigma_{i-1})$. We have that $\text{Sub}(a^+, \Sigma_i) = (N, G)$ where

$$\begin{aligned} N &= \{a^+\} \cup \{b^+ \mid b \in \text{pos}(r)\} \cup \{b^- \mid b \in \text{neg}(r)\} \cup \\ &\quad \bigcup_{b \in \text{pos}(r)} NP_b \cup \bigcup_{b \in \text{neg}(r)} NQ_b \end{aligned}$$

and

$$\begin{aligned} E &= \{(a^+, b^+, +) \mid b \in \text{pos}(r)\} \cup \{(a^+, b^-, +) \mid b \in \text{neg}(r)\} \cup \\ &\quad \bigcup_{b \in \text{pos}(r)} EP_b \cup \bigcup_{b \in \text{neg}(r)} EQ_b \end{aligned}$$

From the inductive hypothesis, we have that P_b 's (resp. Q_b 's) are safe off-line

e-graphs of b^+ (resp. b^-) w.r.t. I_{i-1} and A . This implies that G is a (I_i, A) -based e-graph of a^+ . Furthermore, for every $(a^+, e, +) \in E$, $e \in (\Gamma^{i-1}(J))^p$ or $e \in (\Delta^{i-1}(J))^n$. Thus, $(a^+, a^+) \notin E^{*,+}$.

2. for each $a \in \Delta^i(J) \setminus \Delta^{i-1}(J)$, let $G = \text{Sub}(a^-, \Sigma_i) = (N, E)$. From the definition of G , every node in N is reachable from a^- and $\text{support}(e, G)$ is a local consistent explanation of e w.r.t. I_i and A for every $e \in N$. Thus, G is a (I_i, A) -based e-graph of a^- . Furthermore, it follows from the definition of Σ_i that there exists no node $e \in N$ such that $e \in (\Gamma^i(J) \setminus \Gamma^{i-1}(J))^+$. Thus, if $c^+ \in N$ and $(c^+, c^+) \in E^{*,+}$ then we have that $\text{Sub}(c^+, \Sigma_{i-1})$ is not safe. This contradicts the fact that it is safe due to the inductive hypothesis. □

Lemma 5.1. Let P be a program, J a p-interpretation, and $A = \mathcal{TA}_P(J)$. The following properties hold:

- For each atom $a \in \Gamma(J)$ (resp. $a \in \Delta(J)$), there exists a *safe* off-line e-graph of a^+ (resp. a^-) w.r.t. J and A ;
- for each atom $a \in J^+ \setminus \Gamma(J)$ (resp. $a \in J^- \setminus \Delta(J)$) there exists an on-line e-graph of a^+ (resp. a^-) w.r.t. J and A .

Proof. The first item follows from the Lemma 8.14. The second item of the lemma is trivial due to the fact that $(\{a^+, \text{assume}\}, \{(a^+, \text{assume}, +)\})$ (resp. $(\{a^-, \text{assume}\}, \{(a^-, \text{assume}, -)\})$) is a (J, A) -based e-graph of a^+ (resp. a^-), and hence, is an on-line e-graph of a^+ (resp. a^-) w.r.t. J and A . □

Lemma 5.2. Let P be a program, J be an interpretation, and M be an answer set such that $J \sqsubseteq M$. For every atom a , if (N, E) is a safe off-line e-graph of a w.r.t. J and A where $A = J^- \cap \mathcal{TA}_P(M)$ then it is an off-line justification of a w.r.t. M and $\mathcal{TA}_P(M)$.

Proof. The result is obvious from the definition of off-line e-graph and from the fact that $J^- \cap \mathcal{TA}_P(M) \subseteq \mathcal{TA}_P(M)$. □

Proposition 4. Let M_0, \dots, M_k be a general complete computation and $S(M_0), \dots, S(M_k)$ be an on-line justification of the computation. Then, for each atom a in M_k , the e-graph of a in $S(M_k)$ is an off-line justification of a w.r.t. M_k and $\mathcal{TA}_P(M_k)$.

Proof. This follows immediately from Lemma 5.2, the construction of the snapshots $S(M_i)$, the fact that $M_i \sqsubseteq M_k$ for every k , and M_k is an answer set of P . □