Justifications and Blocking Sets in a Rule-Based Answer Set Computation*

Christopher Béatrix[1], Claire Lefèvre[2], Laurent Garcia[3], and Igor Stéphan[4]

1 **LERIA, University of Angers, Angers, France**
`beatrix@info.univ-angers.fr`
2 **LERIA, University of Angers, Angers, France**
`claire@info.univ-angers.fr`
3 **LERIA, University of Angers, Angers, France**
`garcia@info.univ-angers.fr`
4 **LERIA, University of Angers, Angers, France**
`stephan@info.univ-angers.fr`

Abstract

Notions of justifications for logic programs under answer set semantics have been recently studied for atom-based approaches or argumentation approaches. The paper addresses the question in a rule-based answer set computation: the search algorithm does not guess on the truth or falsity of an atom but on the application or non application of a non monotonic rule. In this view, justifications are sets of ground rules with particular properties. Properties of these justifications are established; in particular the notion of blocking set (a reason incompatible with an answer set) is defined, that permits to explain computation failures. Backjumping, learning, debugging and explanations are possible applications.

1998 ACM Subject Classification D.1.6 Logic Programming, I.2.3 Logic Programming

Keywords and phrases Answer Set Programming, Justification, Rule-based Computation

Digital Object Identifier 10.4230/OASIcs.ICLP.2016.6

1 Introduction

Answer Set Programming (ASP) is a very convenient paradigm to represent knowledge in Artificial Intelligence and to encode Constraint Satisfaction Problems. It is also a very interesting way to practically solve them since some efficient solvers are available [18, 12, 5]. Usually, knowledge representation in ASP is done by means of first-order rules. But, most of the ASP solvers are propositional and they begin by an instantiation phase in order to obtain a propositional program from the first-order one. Furthermore, most of the ASP solvers are based on search algorithms where a choice point is on whether an atom is or is not in a model. But some other solvers like `Gasp` [15], `ASPeRiX` [10, 11] and `OMiGA` [3] are based on principles which do not need this preliminary instantiation of the first-order program: the rule guided approach. The choice point of the search algorithm is on the application or the non application of an on-the-fly instantiated rule.

Justifications in logic programs are intended to provide information about the reason why some property is true, in general why an atom is or is not part of an answer set. The

* This work was supported by ANR (National Research Agency), project ASPIQ under the reference ANR-12-BS02-0003.

© Christopher Béatrix, Claire Lefèvre, Laurent Garcia, and Igor Stéphan;
licensed under Creative Commons License CC-BY
Technical Communications of the 32nd International Conference on Logic Programming (ICLP 2016).
Editors: Manuel Carro, Andy King, Neda Saeedloei, and Marina De Vos; Article No. 6; pp. 6:1–6:15

Open Access Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

main applications are to help users in understanding the program behavior and debugging it. Indeed, in diagnosis or decision systems, it can be important to understand why a decision is made or why a potential decision is not reached; or, in a debugging perspective, explain why an unintended solution is reached or why a given interpretation is not an answer set. Each related work addresses the problem from a specific viewpoint: for example, an atom-based approach using well-founded semantics for [16] and an argumentation framework with attacks and supports for [17]. [16] distinguishes *off-line justification* which is a reason for the truth value of an atom w.r.t. a given answer set (a complete interpretation) from *on-line justification* which is a reason for the truth value of an atom during the computation of an answer set and thus w.r.t. an incomplete interpretation.

The present work deals with on-line justification from a rule-based perspective: a justification is a set of rules with specific status, and truth values of atoms can remain undefined until the end of the computation. In on-line justifications, an interesting question is that of the failure of a computation. In practice, to explain the failures allows to help guide the search and can have direct applications in backjumping and learning. But justifications are interesting by themselves to explain (partial) results of a computation and to debug logic programs.

A rule based computation is a forward chaining process that builds a 3-valued interpretation $\langle IN, OUT \rangle$ in which each atom can be true (belongs to IN), false (belongs to OUT) or undefined (belongs neither to IN nor to OUT). At each revision step, a ground rule is chosen to be applied or to be blocked. During this process, some “reasons” (sets of ground rules, each of them having some properties – “status” – w.r.t. the interpretation under construction) can be associated to atoms justifying their adding to the interpretation. From these first reasons, we are able to compute why an atom is undefined or why the computation fails. A blocking set is defined as a reason that justifies the failure of a computation: it is composed of the applied and blocked rules responsible of the failure. In practice, the blocking sets allow to prune the search tree.

There exist several works on justification [16, 17, 1] and debugging [6, 4, 2]. Papers about justification focus on the reason why some interpretation is an answer set (explanation of the truth values of atoms in an interpretation). To our knowledge, the two closest works are [16] and [17]. [16] encodes an explanation by a graph where the nodes are atoms (annotated “true” or “false”). Their justification is based on the well-founded semantics which determines negative atoms that can be “assumed” to be false (as opposed to atoms which are always false). This corresponds to the `Smodels` solving process. The approach of [17] is in argumentative terms. The ASP program is translated into a theory of argumentation. This allows the construction of arguments and attack relation on these arguments from which justifications can be computed. Justifications are also encoded by graphs: an attack tree of an argument is a graph where the nodes are arguments and the edges are attacks or supports between arguments. An argument-based justification is defined as a flattened version of the preceding tree: it is a set of support and attack relations between atoms. [1] proposes a construction of propositional formulas that encode provenance information for the logic program; justifications can then be extracted from these formulas.

On the other hand, the goal of the debugging systems is to explain why some interpretation is not an answer set, or why an interpretation, expected not to be an answer set, is an answer set and, eventually, to propose repairs of the program. [4] characterize inconsistency in terms of bridge rules: rules which need to be altered for restoring consistency, or combination of rules which causes inconsistency. [6] uses meta-programming technique in order to find semantic errors of programs, based on some types of errors. [2] also uses meta-programming

method to generate propositional formulas that encode provenance information and suggest repairs.

The paper is organized as follows. Section 2 gives some background about ASP. Section 3 presents the concept of *computation*: a constructive characterization of answer sets. In Section 4, notions of justifications and blocking sets are defined, and some of their properties are established. Section 5 concludes by some perspectives.

2 Answer Set Programming

A *normal logic program* is a set of rules like

$$c \leftarrow a_1, \ldots, a_n, not\ b_1, \ldots, not\ b_m. \quad (n \geq 0, m \geq 0)$$

where $c, a_1, \ldots, a_n, b_1, \ldots, b_m$ are atoms built from predicate symbols, constants, variables and function symbols. For a rule r (or by extension for a rule set), we note $head(r) = c$ its *head*, $body^+(r) = \{a_1, \ldots, a_n\}$ its *positive body*, $body^-(r) = \{b_1, \ldots, b_m\}$ its *negative body* and $body(r) = body^+(r) \cup body^-(r)$. When the negative body of a rule is not empty we say that this rule is *non-monotonic*. A *ground substitution* is a mapping from the set of variables to the set of the ground terms (terms without any variable). If t is a term (resp. a an atom) and σ a ground substitution, $\sigma(t)$ (resp. $\sigma(a)$) is a *ground instance* of t (resp. a). A program P can be seen as an intensional version of the propositional program $ground(P) = \bigcup_{r \in P} ground(r)$ where $ground(r)$ is the set of all fully instantiated rules that can be obtained by substituting every variable in r by every constant of the Herbrand universe of P. The set of *generating rules* [8] of an atom set X for a program P, $GR_P(X)$, is defined as $GR_P(X) = \{\sigma(r) \mid r \in P, \sigma \text{ is a ground substitution s.t. } \sigma(body^+(r)) \subseteq X \text{ and } \sigma(body^-(r)) \cap X = \emptyset\}$. A set of ground rules R is *grounded* if there exists an enumeration $\langle r_1 \ldots r_n \rangle$ of the rules of R such that $\forall i \in [1..n]$, $body^+(r_i) \subseteq head\{r_j \mid j < i\}$. Then, X is an answer set of P (originally called a *stable model* [7]) if and only if $X = head(GR_P(X))$ and $GR_P(X)$ is grounded.

3 Rule-based Answer Set Computation

In this section, a constructive characterization of answer sets for normal logic programs, based on a concept of ***ASPeRiX*** *computation* [9], is presented. This concept is itself based on an abstract notion of *computation* for ground programs proposed in [13]. The only syntactic restriction required is that every rule of a program must be *safe*. That is, all variables occurring in the rule occur also in its positive body. Moreover, every constraint (i.e. headless rule) is considered given with the particular head $\perp$ and is also safe.

An ***ASPeRiX*** *computation* for a program P is defined as a process on a computation state based on a *partial interpretation* which is a pair $\langle IN, OUT \rangle$ of disjoint atom sets included in the Herbrand base of P. Intuitively, all atoms in IN belong to a search answer set and all atoms in OUT do not. The notion of partial interpretation defines different status for rules. If r is a rule, σ is a ground substitution and $I = \langle IN, OUT \rangle$ is a partial interpretation: $\sigma(r)$ is *supported* w.r.t. I when $body^+(\sigma(r)) \subseteq IN$, $\sigma(r)$ is *blocked* w.r.t. I when $body^-(\sigma(r)) \cap IN \neq \emptyset$, $\sigma(r)$ is *unblocked* w.r.t. I when $body^-(\sigma(r)) \subseteq OUT$, and r is *applicable* with σ w.r.t. I when $\sigma(r)$ is supported and not blocked.[1]

[1] The negation of blocked, *not blocked*, is different from *unblocked*.

An ASPeRiX computation is a forward chaining process that instantiates and fires one unique rule at each iteration according to two kinds of inference: a monotonic step of *propagation* and a nonmonotonic step of *choice*. To fire a rule means to add the head of the rule in the set IN. If P is a set of first order rules, I is a partial interpretation and R is a set of ground rules: $\Delta_{pro}(P, I, R) = \{(r, \sigma) \mid r \in P, \sigma$ is a ground substitution s.t. $\sigma(r)$ is supported and unblocked w.r.t. I, and $\sigma(r) \notin R\}$, $\Delta_{cho}(P, I, R) = \{(r, \sigma) \mid r \in P, \sigma$ is a ground substitution s.t. $\sigma(r)$ is applicable w.r.t. I and $\sigma(r) \notin R\}$. These sets contain pairs (r, σ) but, for simplicity, we sometimes consider they contain ground rules $\sigma(r)$. They are used in the following definition of an ASPeRiX computation. The specific case of constraints (rules with $\bot$ as head) is treated by adding $\bot$ to OUT set. By this way, if a constraint is fired (violated), $\bot$ should be added to IN and thus, $\langle IN, OUT\rangle$ would not be a partial interpretation. The sets R^{app} and R^{excl} represent the ground rules that are respectively fired and excluded during the computation.

▶ **Definition 1** (ASPeRiX **Computation).** Let P be a first order normal logic program. An *ASPeRiX computation* for P is a sequence $\langle R_i, K_i, I_i\rangle_{i=0}^{\infty}$ of ground rule sets pairs $R_i = \langle R_i^{app}, R_i^{excl}\rangle$, ground rule sets K_i and partial interpretations $I_i = \langle IN_i, OUT_i\rangle$ that satisfies the following conditions:

- $R_0 = \langle\emptyset, \emptyset\rangle$, $K_0 = \emptyset$ and $I_0 = \langle\emptyset, \{\bot\}\rangle$,
- (Revision) 4 possible cases:
  (Propagation) $r_i = \sigma(r)$ for $(r, \sigma) \in \Delta_{pro}(P, I_{i-1}, R_{i-1}^{app})$,
  $R_i = \langle R_{i-1}^{app} \cup \{r_i\}, R_{i-1}^{excl}\rangle$, $K_i = K_{i-1}$
  and $I_i = \langle IN_{i-1} \cup \{head(r_i)\}, OUT_{i-1}\rangle$
  or (Rule choice) $\Delta_{pro}(P \cup K_{i-1}, I_{i-1}, R_{i-1}^{app}) = \emptyset$,
  $r_i = \sigma(r)$ for $(r, \sigma) \in \Delta_{cho}(P, I_{i-1}, R_{i-1}^{app} \cup R_{i-1}^{excl})$,
  $R_i = \langle R_{i-1}^{app} \cup \{r_i\}, R_{i-1}^{excl}\rangle$, $K_i = K_{i-1}$
  and $I_i = \langle IN_{i-1} \cup \{head(r_i)\}, OUT_{i-1} \cup body^-(r_i)\rangle$
  or (Rule exclusion) $\Delta_{pro}(P \cup K_{i-1}, I_{i-1}, R_{i-1}^{app}) = \emptyset$,
  $r_i = \sigma(r)$ for $(r, \sigma) \in \Delta_{cho}(P, I_{i-1}, R_{i-1}^{app} \cup R_{i-1}^{excl})$,
  $R_i = \langle R_{i-1}^{app}, R_{i-1}^{excl} \cup \{r_i\}\rangle$, $K_i = K_{i-1} \cup \{\bot \leftarrow \bigcup_{b \in body^-(r_i)} not\ b.\}$
  and $I_i = I_{i-1}$
  or (Stability) $R_i = R_{i-1}$, $K_i = K_{i-1}$ and $I_i = I_{i-1}$,

If $\exists i \geq 0$, $\Delta_{cho}(P \cup K_i, I_i, R_i^{app} \cup R_i^{excl}) = \emptyset$, then the computation is said to *converge*[2] to the set $IN_\infty = \bigcup_{i=0}^{\infty} IN_i$.

Revision by (Rule exclusion) is not necessary to characterize answer sets. It adds the possibility to block a rule from Δ_{cho} instead of firing it. To block a rule is to add a constraint with the negative atoms of the rule body. This possibility restricts rule choice in Δ_{cho} and thus forbids some computations: if a ground rule r is blocked, the computation can only converge to an answer set whose generating rules do not contain r. It is only useful for having a correspondence between theoretical computations and practical search trees.

▶ **Example 2.** Let P_2 be the following program:
$R_1 : v(1).\quad R_2 : v(2).\quad R_3 : v(3).\quad R_4 : green(4).\quad R_5 : edge(1, 3).\quad R_6 : edge(3, 4).$
$R_7 : red(X) \leftarrow v(X), not\ green(X).\quad R_9 : \bot \leftarrow edge(X, Y), red(X), red(Y).$
$R_8 : green(X) \leftarrow v(X), not\ red(X).\quad R_{10} : \bot \leftarrow edge(X, Y), green(X), green(Y).$

[2] Convergence is not always ensured due to function symbols. The problem can be fixed by limiting the nesting of function symbols.

The following sequence is an `ASPeRiX` computation for P_2:

$I_0 = \langle \emptyset, \{\bot\} \rangle$
(Propagation)
$r_1 = v(1).$ with $(R_1, \emptyset) \in \Delta_{pro}(P_2, I_0, \emptyset)$
$I_1 = \langle \{\mathbf{v(1)}\}, \{\bot\} \rangle$
Steps 2 to 6 are similar: facts of the program are added to IN set by propagation.
$I_6 = \langle \{v(1), v(2), v(3), green(4), edge(1,3), edge(3,4)\}, \{\bot\} \rangle$
(Rule choice) $\Delta_{pro}(P_2, I_6, \{r_1, \cdots, r_6\}) = \emptyset$
$r_7 = red(1) \leftarrow v(1), not\ green(1).$ with $(R_7, \{X \leftarrow 1\}) \in \Delta_{cho}(P_2, I_6, \{r_1, \cdots, r_6\})$
$I_7 = \langle \{v(1), v(2), v(3), \mathbf{red(1)}, green(4), edge(1,3), edge(3,4)\}, \{\bot, \mathbf{green(1)}\} \rangle$
(Rule choice) $\Delta_{pro}(P_2, I_7, \{r_1, \cdots, r_7\}) = \emptyset$
$r_8 = red(2) \leftarrow v(2), not\ green(2).$ with $(R_7, \{X \leftarrow 2\}) \in \Delta_{cho}(P_2, I_7, \{r_1, \cdots, r_7\})$
$I_8 = \langle \{v(1), v(2), v(3), red(1), \mathbf{red(2)}, green(4), edge(1,3), edge(3,4)\},$
$\{\bot, green(1), \mathbf{green(2)}\} \rangle$
(Rule choice) $\Delta_{pro}(P_2, I_8, \{r_1, \cdots, r_8\}) = \emptyset$
$r_9 = red(3) \leftarrow v(3), not\ green(3).$ with $(R_7, \{X \leftarrow 3\}) \in \Delta_{cho}(P_2, I_8, \{r_1, \cdots, r_8\})$
$I_9 = \langle \{v(1), v(2), v(3), red(1), red(2), \mathbf{red(3)}, green(4), edge(1,3), edge(3,4)\},$
$\{\bot, green(1), green(2), \mathbf{green(3)}\} \rangle$

$(R_9, \{X \leftarrow 1, Y \leftarrow 3\}) \in \Delta_{pro}(P_2, I_9, \{r_1, \cdots, r_9\})$ but $r_{10} = (\bot \leftarrow edge(1,3), red(1), red(3).)$ cannot be applied by (Propagation) because its head, $\bot$, is already into the OUT set. The computation does not converge, it is "blocked": the only possible revision is stability.

If the rule $r_7 = (green(1) \leftarrow v(1), not\ red(1).)$ instead of $(red(1) \leftarrow v(1), not\ green(1).)$ with $(R_8, \{X \leftarrow 1\}) \in \Delta_{cho}(P_2, I_6, \{r_1, \cdots, r_6\})$ has been chosen at step 7, other steps being the same, then

$I_9 = \langle \{v(1), v(2), v(3), green(1), red(2), red(3), green(4), edge(1,3), edge(3,4), red(1)\},$
$\{\bot, green(2), green(3)\} \rangle$

$\Delta_{pro}(P_2, I_9, \{r_1, \cdots, r_9\}) = \emptyset,\ \Delta_{cho}(P_2, I_9, \{r_1, \cdots, r_9\}) = \emptyset$
$I_{10} = I_9$

This last `ASPeRiX` computation converges to the set $\{v(1), v(2), v(3), green(1), red(2), red(3), green(4), edge(1,3), edge(3,4)\}$ which is an answer set for P_2.

The following theorem establishes a connection between the results of any `ASPeRiX` computation which converges and the answer sets of a normal logic program.

▶ **Theorem 3.** *[9] Let P be a normal logic program and X be an atom set. Then, X is an answer set of P if and only if there is an `ASPeRiX` computation $S = \langle R_i, K_i, I_i \rangle_{i=0}^{\infty}$, $I_i = \langle IN_i, OUT_i \rangle$, for P such that S converges and $IN_\infty = X$.*

Let us note that in order to respect the revision principle of an `ASPeRiX` computation each sequence of partial interpretations must be generated by using the propagation inference based on rules from Δ_{pro} as long as possible before using the choice based on Δ_{cho} in order to fire a nonmonotonic rule. Then, because of the non determinism of the selection of rules from Δ_{cho}, the natural implementation of this approach leads to a usual search tree where, at each node, one has to decide whether or not to fire a rule chosen in Δ_{cho}. Persistence of applicability of the nonmonotonic rule chosen by (Rule choice) to be fired is ensured by adding to the OUT set all ground atoms from its negative body. On the other branch, where the rule is not fired (Rule exclusion), the translation of its negative body into a new constraint ensures that it becomes impossible to find later an answer set in which this rule is not blocked.

4 Justifications and Blocking Sets

In an ASPeRiX computation, now simply called *computation*, a *reason* is a justification of some property. For instance, the property could be that some atom a belongs to IN (resp. OUT), or that a is undetermined (neither into the IN nor into the OUT sets), or that a constraint c belongs to the K set, or that the computation does not converge. A *reason* is defined as a set of numbered ground rules (numbered rules used for (Revision) in a computation). These rules are those responsible of the satisfaction of the property.

4.1 Reasons of atoms and rules

4.1.1 Reasons of the atoms in IN or OUT sets and of the constraints

We define in this section how reasons of atoms and rules are calculated in a computation: reasons why an atom is added to the IN or OUT sets and reasons why a rule is added to a program. In practice, only constraints are added during the search (to the K set) but it is easier to define reasons for all ground rules (included those issued from the initial program). Reasons are defined as follows in a computation $S = \langle R_i, K_i, I_i \rangle_{i=0}^{n}$ for a program P where $I_i = \langle IN_i, OUT_i \rangle$.

Rules from P. To each instance of rule and constraint of the initial program reason $\{r_0\}$ is associated where r_0 is a new constant with number 0. For every rule instance r of the initial program, $reason(r, S) = \{r_0\}$.

(Propagation) step. During the (Propagation) step, the head of a ground rule r_i is added to the IN_i set because the rule is supported and unblocked: all the atoms of the positive body belong to IN_{i-1} and all the atoms of the negative body belong to OUT_{i-1}. The reason of the adding of $head(r_i)$ to the IN_i set is the set of reasons why the atoms of the body are in the partial interpretation plus the rule itself: $reason(head(r_i), S) = \bigcup_{a \in body(r_i)} reason(a, S) \cup \{r_i\}$.

(Rule choice) step. During a (Rule Choice), a rule r_i is chosen to be unblocked and the atoms of the negative body of the rule are added to the OUT_i set with the only justification being that r_i has been arbitrary unblocked: $reason(a, S) = \{r_i\}$, for all $a \in body^-(r_i) \setminus OUT_{i-1}$.

During the same step, the head of r_i is added to the IN_i set (since r_i is henceforth supported and unblocked). The justification is the same as that in (Propagation) step (except that the rule is only non blocked at step $i-1$ and becomes unblocked only at step i): $reason(head(r_i), S) = \bigcup_{a \in body(r_i)} reason(a, S) \cup \{r_i\}$.

(Rule exclusion) step. During a (Rule exclusion) step, where a rule r_i is chosen to be blocked, a constraint is added to the K_i set. To such a constraint, generated to block the chosen rule r_i, is associated the reason $\{r_i\}$ since this arbitrary choice is only justified by itself: $reason(\bot \leftarrow body^-(r_i)., S) = \{r_i\}$.

▶ **Example 4** (Example 2 continued)**.** Reason of instances of rules R_1 to R_{10} is $\{r_0\}$. At propagation step 1, $v(1)$ is added to IN with reason $\{r_1\}$. Then at choice step 7, when the rule $r_7 = (red(1) \leftarrow v(1), not\ green(1).)$ is chosen to be unblocked, $green(1)$ is added to OUT with reason $\{r_7\}$ and $red(1)$ is added to IN with reason $\{r_1, r_7\}$. If the rule r_7 was excluded instead of being chosen, the constraint $(\bot \leftarrow not\ green(1).)$ would be added to the K set with the reason $\{r_7\}$ (cf. Figure 1).

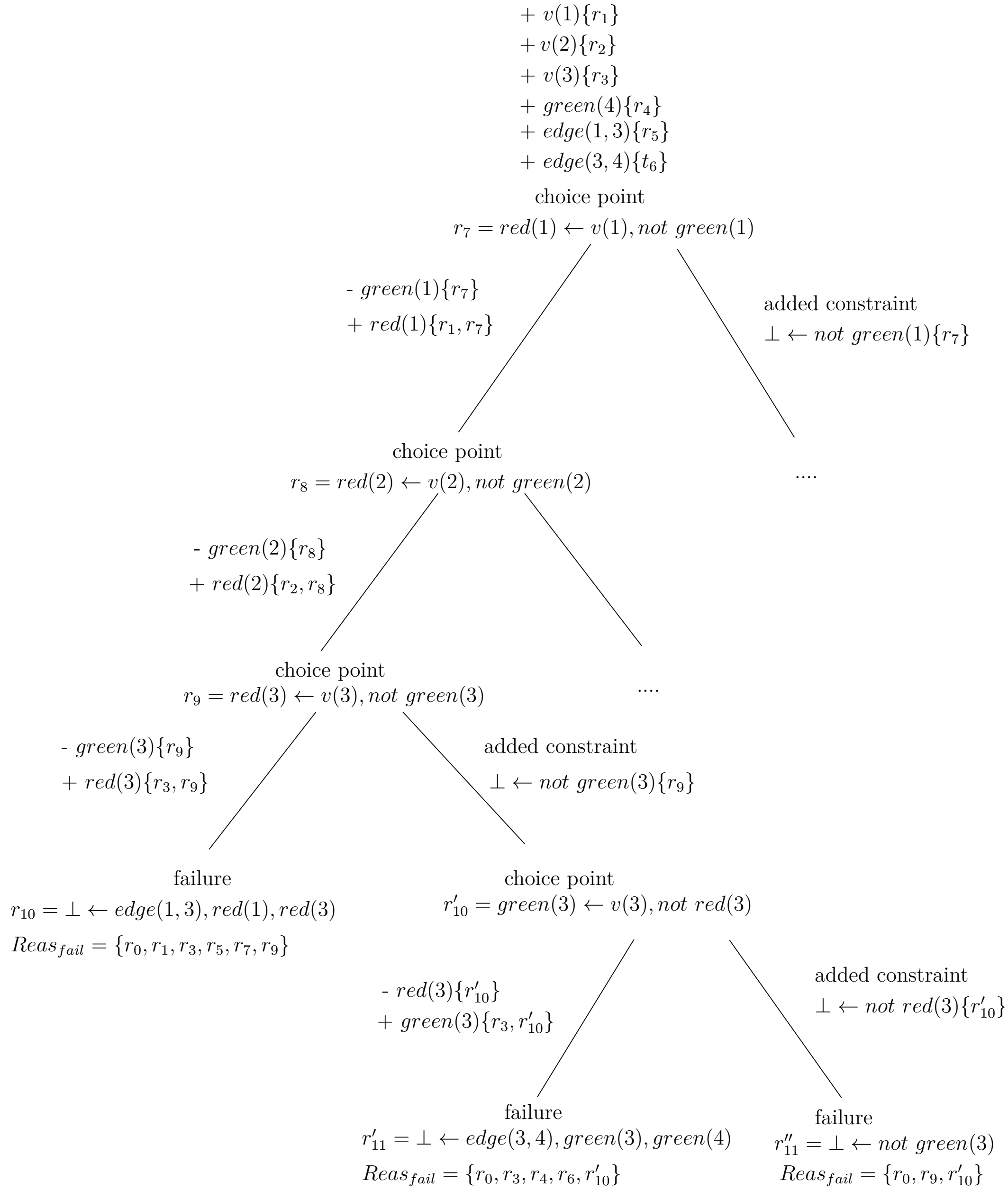

Figure 1 Part of the search tree for the program P_2 of Example 2. At each node, left branch is (Rule choice) and right branch is (Rule exclusion). Each branch corresponds to a computation. Adding atom a to IN set with reason R is symbolized by $+\ a\ R$, and adding a to OUT set with reason R is symbolized by $-\ a\ R$.

4.1.2 Reasons of the undetermined atoms

In a computation, the interpretation $\langle IN, OUT\rangle$ may remain partial until the end of the sequence. If an atom a_0 is undetermined (i.e., not in the IN nor in the OUT sets), it is only known that a_0 cannot be proven. Intuitively, if a_0 cannot be proven it is because no ground rule concluding a_0 can be fired. Then, it has to be determined why a rule has never been fired along the sequence of a computation.

Let $S = \langle R_i, K_i, I_i\rangle_{i=0}^{k}$ be a computation prefix with $I_i = \langle IN_i, OUT_i\rangle$ and $R_i = \langle R_i^{app}, R_i^{excl}\rangle$, and r be a ground rule which has not been fired during the computation: $r \notin R_k^{app}$. The reason why r has not been fired may be: (i) There is an atom a from its positive body which is in the OUT_k set and then prevents the rule from being supported; or (ii) there is an atom a from its negative body which is in the IN_k set and then blocks the rule. For such an atom a, a reason why the rule r is not applicable is the reason why a belongs to the IN_k set (resp. OUT_k).

Another possible reason why r has not been fired may be that (iii) there is an atom a from its positive body which is undetermined, i.e., it belongs neither to the IN_k set nor to the OUT_k set and, again, prevents the rule from being supported. In this case, a reason why r is not applicable is the reason why a is undetermined.

Finally, if r has not been fired despite it was applicable, it means that the rule has been chosen for (Rule exclusion), and then blocked by adding a constraint to the K set. In this case, the reason why the rule r is not applicable is simply the reason of this constraint, i.e., this arbitrary choice to exclude r.

▶ **Example 5.** Let $I_k = \langle\{x\}, \{c, d\}\rangle$, a be a non provable atom, and $r_1 = (a \leftarrow y,\ not\ c.)$ and $r_2 = (a \leftarrow x,\ not\ b,\ not\ d.)$ be the only two ground rules concluding a.

Atom a is not provable because, firstly, r_1 is not supported (due to undetermined atom y) and thus cannot be fired, and, secondly, r_2 has not been fired despite it is applicable. Then r_2 has necessarily been chosen for (Rule exclusion) and thus blocked by adding the constraint $(\bot \leftarrow not\ b,\ not\ d.)$ to the K set. Finally, a has failed to be proven because the undetermined atom y prevents r_1 from being supported, and $(\bot \leftarrow not\ b,\ not\ d.)$ blocks r_2. The reason of undetermined atom a will be the union of the reason of undetermined atom y in S and $reason(\bot \leftarrow not\ b,\ not\ d., S)$.

In the following definition, a reason of an undetermined atom a is defined with respect to a sequence $T = \langle R_i, Atoms_i, Reas_i\rangle_{i=0}^{\infty}$. The idea is the following. For each i, $Atoms_i$ is the set of undetermined atoms for which a reason has to be defined, it is then initialized with $\{a\}$. For each ground rule r whose head is in $Atoms_i$, we have to determine a reason for which r has not been fired. At each step i, such a rule r_i is chosen, and a reason for which it has not been fired is determined and added to the $Reas_i$ set. If this reason involves another undetermined atom b, b is added to the $Atoms_i$ set. R_i is the set of ground rules already treated at step i, thus the sequence converges when all ground rules whose head is in $Atoms_i$ are treated.

If P is a program and a is an atom, $hrule(a, P) = \{r \in ground(P) \mid head(r) = a\}$.

▶ **Definition 6** (Reason of undetermined atoms). Let P be a program, $S = \langle R_i, K_i, I_i\rangle_{i=0}^{n}$ be a computation prefix with $I_n = \langle IN_n, OUT_n\rangle$, and a be an undetermined atom: $a \notin IN_n \cup OUT_n$. A *reason of undetermined atom a*, denoted $reason_{und}(a, S)$, is defined with respect to a sequence $T = \langle R_i, Atoms_i, Reas_i\rangle_{i=0}^{\infty}$ where for each i, R_i is a set of ground rules, $Atoms_i$ is an atomset, and $Reas_i$ is a set of ground rules, that satisfies the following conditions:

- $R_0 = \emptyset$, $Atoms_0 = \{a\}$, $Reas_0 = \{r_0\}$
- $\forall i > 0$, $r_i \in \bigcup_{at \in Atoms_{i-1}} hrule(at, P) \setminus R_{i-1}$ and satisfies one of the following conditions:
  (i) $\exists l \in (body^+(r_i) \cap OUT_n)$,
  or (ii) $\exists l \in (body^-(r_i) \cap IN_n)$
  Then, $R_i = R_{i-1} \cup \{r_i\}$, $Atoms_i = Atoms_{i-1}$,
  $Reas_i = Reas_{i-1} \cup reason(l, S)$
  or (iii) $\exists l \in (body^+(r_i) \setminus (IN_n \cup OUT_n))$,
  Then, $R_i = R_{i-1} \cup \{r_i\}$, $Atoms_i = Atoms_{i-1} \cup \{l\}$, $Reas_i = Reas_{i-1}$
  or (iv) $\exists const \in K$ such that $const = (\bot \leftarrow \cup_{b \in body^-(r_i)} not\ b.)$
  and $reason(const, S) = \{r_i\}$
  Then, $R_i = R_{i-1} \cup \{r_i\}$, $Atoms_i = Atoms_{i-1}$,
  $Reas_i = Reas_{i-1} \cup reason(const, S)$
  or (v) $R_i = R_{i-1}$, $Atoms_i = Atoms_{i-1}$, $Reas_i = Reas_{i-1}$
- (Convergence) $\exists i \geq 0$, $R_i = \bigcup_{at \in Atoms_{i-1}} hrule(at, P)$

The sequence $T = \langle R_i, Atoms_i, Reas_i \rangle_{i=0}^{\infty}$ is said to converge with $Reas_\infty = \bigcup_{i=0}^{\infty} Reas_i$. Then, $reason_{und}(a, S) = Reas_\infty$.

4.2 Blocking sets

Each rule r from a reason $Reason$ can be of three types according to what justifies r to belong to the reason: it can be into the reason in order to justify the truth of its head ($Reason_S^{decl}$), in this case r has been fired at a (Propagation) or (Rule choice) step and $r \in reason(head(r), S) \subseteq Reason$; or r can be into the reason in order to justify the falsity of an atom from its negative body ($Reason_S^{nblock}$), in this case r has been unblocked at a (Rule choice) step and, for $l \in body^-(r)$, $\{r\} = reason(l, S) \subseteq Reason$; or r can be into the reason in order to justify an undetermined atom ($Reason_S^{block}$), in this case r has been blocked at a (Rule exclusion) step by adding a constraint c and $\{r\} = reason(c, S) \subseteq Reason$.

▶ **Definition 7** (Reason types). Let P be a program, $S = \langle K_i, R_i, I_i \rangle_{i=0}^{n}$ be a computation prefix for P with $R_i = \langle R_i^{app}, R_i^{excl} \rangle$, and $Reason$ be a set of numbered rules.

- $Reason_S^{decl} = \{r_i \in Reason \cap R_n^{app} \mid reason(head(r_i), I_i) \subseteq Reason\}$
- $Reason_S^{nblock} = \{r_i \in Reason \cap R_n^{app} \mid reason(head(r_i), I_i) \nsubseteq Reason\}$
- $Reason_S^{block} = Reason \cap R_n^{excl}$

▶ **Example 8** (Example 4 continued). Consider $Reason = \{r_7\}$, the reason of $green(1)$. $r_7 \in R^{app}$ and $r_7 \in R_S^{nblock}$ because $reason(head(r_7)) = reason(red(1)) = \{r_1, r_7\} \nsubseteq Reason$. But if we consider $Reason = \{r_1, r_7\}$, the reason of $red(1)$, $r_7 \in R^{app}$ and $r_7 \in R_S^{decl}$ because $reason(head(r_7)) \subseteq Reason$. In the first case, r_7 is the reason why the rule is unblocked while in the second case r_7 belongs to the reason because it is fired.

In a computation S which converges to an answer set $X = IN_\infty$, the set of fired rules R^{app} coincides with the generating rules of X, $GR_P(X)$, and $X = head(GR_P(X))$. A reason is a subset of the fired and excluded rules in a computation. For an answer set to be compatible with a reason it must be established that (1) the rules from R^{app} that belong to the reason because they are fired in the computation are generating rules of X, (2) the rules from R^{app} that belong to the reason because they are not blocked in the computation are not blocked w.r.t. X (regardless of whether or not supported w.r.t. X), (3) the rules from R^{excl} that belong to the reason are blocked w.r.t. X (again, regardless of whether or not supported).

▶ **Definition 9** (Compatible). Let P be a program, $S = \langle K_i, R_i, I_i\rangle_{i=0}^{k}$ be a computation prefix for P with $R_k = \langle R_k^{app}, R_k^{excl}\rangle$, and $Reas$ be a set of ground rules such that $Reas \subseteq R_k^{app} \cup R_k^{excl} \cup \{r_0\}$.

An atom set X is *compatible* with $Reas$ if $\begin{cases} Reas_S^{decl} \subseteq GR_P(X) \\ \forall r \in Reas_S^{nblock}, body^-(r) \cap X = \emptyset \\ \forall r \in Reas_S^{block}, body^-(r) \cap X \neq \emptyset \end{cases}$

The rules belonging to a reason are those responsible of the satisfaction of some property. For instance, the property could be that some atom a belongs to IN (resp. OUT), or that a is undetermined, or that the computation does not converge. For a reason to be a real reason of a computation property, each answer set X compatible with the reason must satisfy this property. For instance, Theorem 14 of Section 4.3.1 says that each answer set X compatible with the reason of an undetermined atom a (Def. 6) verifies that $a \notin X$. If there is no answer set compatible with some reason $Reas$, then we say that $Reas$ is a blocking set.

▶ **Definition 10** (Blocking set). Let P be a program, $S = \langle K_i, R_i, I_i\rangle_{i=0}^{k}$ be a computation prefix for P with $R_k = \langle R_k^{app}, R_k^{excl}\rangle$ and $Reas$ be a set of ground rules such that $Reas \subseteq R_k^{app} \cup R_k^{excl} \cup \{r_0\}$. $Reas$ is a *blocking set* if there is no answer set X for P compatible with $Reas$.

▶ **Example 11** (Example 8 continued). Consider the sequence $S = \langle K_i, R_i, I_i\rangle_{i=0}^{9}$ with $R_9^{app} = \{r_1, \dots, r_9\}$. $Reas = \{r_1, r_3, r_5, r_7, r_9\}$ is a blocking set: $Reas \subseteq Reas_S^{decl}$ and there is no answer set X such that $Reas \subseteq GR_P(X)$ since if r_7 and r_9 are generating rules, vertices 1 and 3 are red and the constraint ($\bot \leftarrow edge(1,3), red(1), red(3).$) is a generating rule too. Note that r_0 has no impact on blocking sets and thus $Reas \cup \{r_0\}$ is a blocking set too.

4.3 Failures

In a concrete calculation of answer sets, the search process can be represented by a search tree where the nodes are "guesses" about supported rules to be unblocked (Rule choice) or blocked (Rule exclusion). In such a tree, each branch corresponds to a computation prefix which converges (success branch) or not (failure branch). In case of failure, some backtrack must be done in order to explore another branch. Figure 1 illustrates a part of the search tree of Example 2.

The failures presented here correspond to the computations (branches) that do not converge. The first case, *blocked prefix*, corresponds to branches where no revision is available. The second case, *failure combination*, corresponds to a node where the two branches fail.

4.3.1 Blocked computations

A computation prefix is *blocked* if the computation does not converge and the only possible revision is stability. A computation can be blocked in two cases: either a contradiction is detected by the (Propagation) step (there is a rule r_i which is supported and unblocked but it cannot be fired because its head is already into the OUT set[3]), either there is no more applicable rule but there is at least a non satisfied constraint (i.e. supported and not blocked). Note that all other applicable rules can be chosen by (Rule choice) or, if the head is already in the OUT set, by (Rule exclusion). Thus, a computation cannot be blocked due to an applicable rule other than a constraint.

[3] For the computation to be blocked, we impose in Definition 12 that all rules from Δ_{pro} cannot be fired. But in practice it suffices than one rule from Δ_{pro} cannot be fired in order to ensure the failure.

▶ **Definition 12** (Blocked Prefix and Failure Reason). Let P be a program, $S = \langle K_i, R_i, I_i \rangle_{i=0}^k$ be a prefix of a computation for P with $R_i = \langle R_i^{app}, R_i^{excl} \rangle$ and $I_i = \langle IN_i, OUT_i \rangle$. S is said to be *blocked* if S satisfies one of the following conditions. In each case, a *failure reason* due to a ground rule rf, noted $reason_{fail}(S, rf)$, is defined.

(Propagation failure) $\Delta_{pro}(P \cup K_k, I_k, R_k^{app}) \neq \emptyset$
$\forall r \in \Delta_{pro}(P \cup K_k, I_k, R_k^{app}), head(r) \in OUT_k$
$reason_{fail}(S, rf) = \bigcup_{a \in body(rf)} reason(a, S) \;\cup\; reason(rf, S) \;\cup\; reason(head(rf), S)$
with $rf \in \Delta_{pro}(P \cup K_k, I_k, R_k^{app})$

or (Non satisfied constraint) $\Delta_{pro}(P \cup K_k, I_k, R_k^{app}) = \emptyset$,
$\Delta_{cho}(P, I_k, R_k^{app} \cup R_k^{excl}) = \emptyset$, $\Delta_{cho}(K_k, I_k, R_k^{app} \cup R_k^{excl}) \neq \emptyset$
$reason_{fail}(S, rf) = reason(rf, S) \cup$
$\bigcup_{a \in (body^-(rf) \cap OUT_k)} reason(a, S) \;\cup\; \bigcup_{a \in (body^-(rf) \setminus OUT_k)} reason_{und}(a, S)$
with $rf \in \Delta_{cho}(K_k, I_k, R_k^{app} \cup R_k^{excl})$

In the first case, there exists at least a rule $rf \in \Delta_{pro}(P \cup K_k, I_k, R_k^{app})$ with $head(r) \in OUT_k$, the reason of the contradiction is the reason why rf is supported and unblocked, the reason of the rule itself (which is $\{r_0\}$ if it is not a constraint added by (Rule Exclusion)) and the reason why $head(rf)$ is in the OUT_k set.

In the second case, there exists at least a non satisfied constraint c, the reason of this failure is the set of reasons which make the constraint not blocked (such a constraint from K set has an empty positive body). Note that the constraint is not blocked ($body^-(c) \cap IN_k = \emptyset$) but not unblocked ($body^-(c) \not\subseteq OUT_k$) otherwise it would have been fired during the propagation step. Hence, there is at least an atom in the negative body whose status remained undetermined.

▶ **Example 13** (Example 2 continued). The sequence $S = \langle K_i, R_i, I_i \rangle_{i=0}^9$ is a blocked prefix (see the left most branch of the tree of Figure 1): $(R_9, \{X \leftarrow 1, Y \leftarrow 3\}) \in \Delta_{pro}(P_2, I_9, \{r_1, \cdots, r_9\})$ and $r_{10} = (\bot \leftarrow edge(1,3), red(1), red(3).)$ but $head(r_{10}) = \bot \in OUT$. $reason_{fail}(S, r_{10}) = reason(edge(1,3), S) \cup reason(red(1), S) \cup reason(red(3), S) \cup reason(r_{10}, S) \cup reason(\bot, S) = \{r_0, r_1, r_3, r_5, r_7, r_9\}$.

Properties of a blocked computation prefix can then be established. The following theorem says that a reason RU of an undetermined atom a (see Def. 6) is a real justification of the non provability of a: a cannot belong to an answer set compatible with RU.

▶ **Theorem 14.** *Let P be a program, $S = \langle K_i, R_i, I_i \rangle_{i=0}^k$ be a blocked computation prefix for P, a be an atom such that $a \notin (IN_k \cup OUT_k)$, and RU be a reason of the undetermined atom a. For all answer set X compatible with RU, $a \notin X$.*

▶ **Example 15.** Consider the branch leading to the third leaf of the tree of Figure 2. $green(3)$ is undetermined because of r'_{10} (Rule exclusion), and $reason_{und}(green(3)) = \{r_0, r'_{10}\}$. Theorem 14 guarantees that for all answer set X such that r'_{10} is blocked, $green(3) \notin X$.

The failure reason RF of a blocked computation prefix is also a real justification of failure: there is no answer set compatible with RF.

▶ **Theorem 16.** *Let P be a program, S be a blocked prefix of a computation and RF be a failure reason for S. RF is a blocking set.*

▶ **Example 17** (Example 13 continued). $reason_{fail}(S, r_{10}) = \{r_0, r_1, r_3, r_5, r_7, r_9\}$ is a blocking set (see Example 11). This means that the corresponding steps of the computation are those responsible of the failure. Other revision steps, for instance the (Propagation) $r_4 = (green(4).)$ or the (Rule choice) $r_8 = (red(2) \leftarrow v(2), not\ green(2).)$ have nothing to do with this failure.

4.3.2 Failure combination

We call *choice points* the steps of a computation S where (Rule choice) or (Rule exclusion) are used. In practice, they correspond to nodes in a search tree. If r_i and r_j are numbered rules, $r_i < r_j$ iff $i < j$. If R is a set of numbered rules, $max(R) = r$ iff for all $r_i \in R, r_i \leq r$, and $R_{<r_i} = \{r \in R \mid r < r_i\}$. If P is a program, $S = \langle K_i, R_i, I_i\rangle_{i=0}^{n}$ is a computation prefix for P with $R_i = \langle R_i^{app}, R_i^{excl}\rangle$, and *Reason* is a set of numbered rules, $choicePoints(S) = \{r_i \in R_i^{app} \cup R_i^{excl} \mid i \in [1 \dots n], \Delta_{pro}(P \cup K_{i-1}, I_{i-1}, R_{i-1}^{app}) = \emptyset\}$ are the ground rules r_i used for choice points in S, $choicePoints(Reason, S) = Reason \cap choicePoints(S)$ is the restriction of the preceding set to the rules belonging to some reason *Reason* and $lastChoicePoint(Reason, S) = max(choicePoints(Reason, S))$ is the last rule (the rule with the greatest number) from *Reason* used for a choice point in S.

Suppose two computations that do not converge and that are the same up to a choice point lc. At this step, the computations differ: one uses rule r_{lc} for (Rule choice) and the other use the same rule for (Rule exclusion). If r_{lc} is the last choice point involved in the two failure reasons, then we can conclude that this rule r_{lc} is not implicated in the failure: the rule can be applied or excluded (thus, to be a generating rule or not), the computation fails in both cases. So, the failure exists prior to this choice point lc. A new failure reason can be defined by joining the two failure reasons and restricting them to the rules involved in the computation at a step preceding lc. If the greatest numbered rule of this new failure reason is r_k, then we can conclude that the computation prefix ending at step k will fail too: it is a failure prefix.

▶ **Definition 18** (Failure Prefix and Failure Reason). Let P be a program. A *failure prefix* of a computation prefix for P and a failure reason for this prefix are defined as follows:

1. A blocked prefix is a failure prefix and its failure reason is defined as in Definition 12.
2. Let $S_1 = \langle K_{1i}, R_{1i}, I_{1i}\rangle_{i=0}^{n}$ and $S_2 = \langle K_{2i}, R_{2i}, I_{2i}\rangle_{i=0}^{m}$ be two failure prefix of computation for P with $R_{1i} = \langle {R_1}_i^{app}, {R_1}_i^{excl}\rangle$ and $R_{2i} = \langle {R_2}_i^{app}, {R_2}_i^{excl}\rangle$ and let RF_1 and RF_2 be two failure reasons for, respectively, S_1 and S_2 such that:
   - $lastChoicePoint(RF_1, S_1) = lastChoicePoint(RF_2, S_2) = r_{lc}$
   - $\langle S_1\rangle_{i=0}^{lc-1} = \langle S_2\rangle_{i=0}^{lc-1}$
   - $r_{lc} \in {R_1}_{lc}^{app}$ and $r_{lc} \in {R_2}_{lc}^{excl}$

   Let $reasonFail = {RF_1}_{<r_{lc}} \cup {RF_2}_{<r_{lc}}$ and $r_k = max(reasonFail)$.
   $\langle S_1\rangle_{i=0}^{k} = \langle S_2\rangle_{i=0}^{k}$ is a failure prefix and $reasonFail$ is a *failure reason* for this prefix.

The following theorem establishes that a failure reason of a failure prefix is a real justification of failure: no answer set is compatible with it. A corollary is that a failure prefix cannot be extended to a computation which converges.

▶ **Theorem 19.** *Let P be a program, S be a failure prefix of a computation and RF be a failure reason for S. Then, RF is a blocking set.*

▶ **Theorem 20.** *Let P be a program and S be a failure prefix of a computation for P. For any computation S' with prefix S, S' does not converge.*

▶ **Example 21** (Example 2 continued). Suppose that at step 9 (cf. Fig. 1), the rule r_9 is excluded instead of being chosen. Then the constraint ($\bot \leftarrow$ *not* $green(3)$.) is added to K_9 with the reason $\{r_9\}$. Step 10 can be the choice of the rule $r'_{10} = (green(3) \leftarrow v(3), not\ red(3).)$; in this case $r'_{11} = (\bot \leftarrow edge(3,4), green(3), green(4).) \in \Delta_{pro}(P_2 \cup K_{10}, I_{10}, \{r_1, \cdots, r_{10}\})$ and $reason_{fail}(S', r'_{11}) = \{r_0, r_3, r_4, r_6, r'_{10}\}$. If r'_{10} is excluded at step 10, the constraint ($\bot \leftarrow$ *not* $red(3)$.) is added to K_{10} with the reason $\{r'_{10}\}$. There are

non satisfied constraints, for instance $r''_{11} = (\bot \leftarrow \textit{not green}(3).)$. $\textit{green}(3)$ is undefined with reason $\{r_0, r'_{10}\}$ and $\textit{reason}_{fail}(S'', r''_{11}) = \{r_0, r_9, r'_{10}\}$. The combination of the two failures leads to $\textit{reason}_{fail}(S', r'_{10}) = \{r_0, r_3, r_4, r_6, r_9\}$. At the preceding choice point (r_9), a new combination of failures leads to $\textit{reason}_{fail}(S, r_9) = \{r_0, r_1, r_3, r_4, r_5, r_6, r_7\}$.

4.3.3 Application to Backjumping

Our approach has already been used for backjumping in the solver `ASPeRiX`: failure reasons are computed during the solving process and permit to jump to the last choice point related to the failure instead of a simple chronological backtrack.

The nodes of the search tree correspond to choice points where an instantiated rule is chosen to be applied (left branch) or to be blocked (right branch). In case of failure, a reason of the failure is computed in order to know the nodes implicated in the failure and to avoid visiting sub-trees where the same failure will necessarily occur again. A failure on a leaf of the tree correspond to a blocked prefix of a computation (see Definition 12 and Theorem 16). When the left and right branches of a node both fail, their failure reasons can be combined to determine the reason of the failure of the sub-tree. This corresponds to a failure prefix of a computation (see Definition 18 and Theorem 19). When the combination of failures permits backjumping, Theorem 20 guarantees that the jumped sub-trees cannot lead to an answer set. For instance at step 9 of Figure 1, $\textit{reason}_{fail}(S, r_9) = \{r_0, r_1, r_3, r_4, r_5, r_6, r_7\}$ (see Example 21), the choice point 8 is not responsible of the failure and the right branch of this node can be safely jumped.

In practice the reason defined above, constituted by the ground rules responsible of the failure, are too detailed. To each rule is then associated the node (choice point) in the tree where it is used. So the computed reasons are only sets of choice points (instead of sets of rules), and suffice to know the last choice responsible of the failure. For instance, the above failure reason will become $\{0, 7\}$ where 0 represents all revisions done before the first choice point.

The implementation is only a prototype that uses Prolog to build the ground rules necessary to compute the reason of undetermined atoms. Indeed, the set of ground rules whose head is a given atom cannot be easily computed in our approach where the rules are first order ones. The backward chaining of Prolog can solve the problem, although not very effectively. But it shows a drastic reduction of the number of choice points for some programs. This will be developed in another paper.

5 Conclusion

Notions of justifications and blocking sets for rule-based answer set computations have been presented with theorems establishing that justifications are "true" ones. These justifications meet those of other related approaches. Each of them views justifications in a particular perspective. Our viewpoint is that of a rule-based computation where a reason is a set of ground rules with particular properties. Our justifications are comparable to on-line justifications of [16]: they can be defined during the computation process. A difference is that we define justifications for undetermined atoms; in atom-based approaches, all atoms are determined at the end of a computation.

These justifications have already been used for backjumping in the solver `ASPeRiX` Other direct applications are learning, interactive debugging and explanation of answers in diagnosis systems. A preliminary approach on learning in a rule-based system is proposed in [19]. An important problem is that learned rules are generally constraints and that constraints are

not used for propagation in rule-based solvers. So progress must be done for better treating constraints and/or learning rules that are not constraints. For debugging, the rule-based approach seems interesting because it allows stepping the search of answer sets. For instance, [14] proposes such an approach of stepping with a notion of computation closed to ours. Blocking sets could also explain absence of solution and help to propose repairs.

References

1 C. V. Damásio, A. Analyti, and G. Antoniou. Justifications for logic programming. In *LPNMR 2013,*, pages 530–542, 2013.

2 C. V. Damásio, J. Moura, and A. Analyti. Unifying justifications and debugging for answer-set programs. In *ICLP 2015*, 2015.

3 M. Dao-Tran, T. Eiter, M. Fink, G. Weidinger, and A. Weinzierl. OMiGA: An open minded grounding on-the-fly answer set solver. In *JELIA 2012*, pages 480–483, 2012.

4 T. Eiter, M. Fink, P. Schüller, and A. Weinzierl. Finding explanations of inconsistency in multi-context systems. In *KR 2010*, 2010.

5 M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-driven answer set solving. In *IJCAI 2007*, pages 386–392, 2007.

6 M. Gebser, J. Pührer, T. Schaub, and H. Tompits. A meta-programming technique for debugging answer-set programs. In *AAAI 2008*, pages 448–453, 2008.

7 M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Logic Programming, Proceedings of the Fifth International Conference and Symposium*, pages 1070–1080, 1988.

8 K. Konczak, T. Linke, and T. Schaub. Graphs and colorings for answer set programming. *Theory and Practice of Logic Programming*, 6:61–106, 1 2006. doi:10.1017/S1471068405002528.

9 C. Lefèvre, C. Béatrix, I. Stéphan, and L. Garcia. Asperix, a first order forward chaining approach for answer set computing. *CoRR*, abs/1503.07717:(to appear in TPLP), 2015. URL: http://arxiv.org/abs/1503.07717.

10 C. Lefèvre and P. Nicolas. A first order forward chaining approach for answer set computing. In *LPNMR 2009*, pages 196–208, 2009.

11 C. Lefèvre and P. Nicolas. The first version of a new ASP solver : ASPeRiX. In *LPNMR 2009*, pages 522–527, 2009.

12 N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006. doi:10.1145/1149114.1149117.

13 L. Liu, E. Pontelli, T. C. Son, and M. Truszczynski. Logic programs with abstract constraint atoms: The role of computations. *Artificial Intelligence*, 174(3-4):295–315, 2010. doi:10.1016/j.artint.2009.11.016.

14 J. Oetsch, J. Pührer, and H. Tompits. Stepping through an answer-set program. In *LPNMR 2011*, pages 134–147, 2011.

15 A. Dal Palù, A. Dovier, E. Pontelli, and G. Rossi. Answer set programming with constraints using lazy grounding. In *ICLP 2009*, 2009.

16 E. Pontelli, T. C. Son, and O. El-Khatib. Justifications for logic programs under answer set semantics. *Theory and Practice of Logic Programming*, 9(1):1–56, 2009. doi:10.1017/S1471068408003633.

17 C. Schulz and F. Toni. Justifying answer sets using argumentation. *Theory and Practice of Logic Programming*, 16(1):59–110, 2016. doi:10.1017/S1471068414000702.

18 P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234, 2002. doi:10.1016/S0004-3702(02)00187-X.

19 A. Weinzierl. Learning non-ground rules for answer-set solving. In *2nd Workshop on Grounding and Transformations for Theories with Variables, GTTV 2013*, 2013.