



HAL
open science

A formalism for causal explanations with an Answer Set Programming translation

Yves Moinard

► **To cite this version:**

Yves Moinard. A formalism for causal explanations with an Answer Set Programming translation. 4th International Conference on Knowledge Science, Engineering & Management (KSEM 2010), Sep 2010, Belfast, United Kingdom. inria-00511093

HAL Id: inria-00511093

<https://inria.hal.science/inria-00511093>

Submitted on 23 Aug 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A formalism for causal explanations with an Answer Set Programming translation

HAL version, Aug 23, 2010 see LNCS (KSEM 2010) for conference version.

Yves Moinard

INRIA Bretagne Atlantique, IRISA, Campus de Beaulieu, 35042 Rennes cedex, France
email: moinard@irisa.fr

August 23, 2010

Abstract

We examine the practicality for a user of using Answer Set Programming (ASP) for representing logical formalisms. Our example is a formalism aiming at capturing causal explanations from causal information. We show the naturalness and relative efficiency of this translation job. We are interested in the ease for writing an ASP program. Limitations of the earlier systems made that in practice, the “declarative aspect” was more theoretical than practical. We show how recent improvements in working ASP systems facilitate the translation.

1 Introduction

We consider a formalism designed in collaboration with Philippe Besnard and Marie-Odile Cordier, aiming at a logical formalization of explanations from causal and “is-a” statements. Given some information such as “fire causes smoke” and “grey smoke is a smoke”, if “grey smoke” is established, we want to infer that “fire” is a (tentative) explanation for this fact. The formalization [2] is expressed in terms of *rules* such as “if α causes β and γ isa β , then α explains γ provided $\{\alpha, \gamma\}$ is possible”. This concerns looking for paths in a graph and ASP is good for this. There exists efficient systems, such as DLV [6] or clasp[D] (www.dbai.tuwien.ac.at/proj/dlv/ or potassco.sourceforge.net/).

Transforming formal rules into an ASP program is easy. ASP should then be an interesting tool for researchers when designing a new theoretical formalization as the one examined here. When defining a theory, ASP programs should help examining a great number of middle sized examples. Then if middle sized programs could work, a few more optimization techniques could make real sized examples work with the final theory.

In fact, even if ASP allows such direct and efficient translation, a few problems complicate the task. The poor data types available in pure ASP systems is

a real drawback, since our rules involve sets. Part of this difficulty comes from a second drawback: In ASP, it is hard to reuse portions of a program. Similar rules should be written again, in a different way. Also, “brave” or “cautious” reasoning is generally not allowed (except with respect to precise “queries”). In ASP, “the problem is the program” and the “solution” consists in one or several sets of atoms satisfying the problem. Each such set is an *answer set*. Brave and cautious solutions mean to look for atoms true respectively in some or in all the answer sets.

This complicates the use of ASP: any modification becomes complex. However, things are evolving, e.g. DLV-Complex (www.mat.unical.it/dlv-complex), deals with the data structure problem and DLT [3] allows the use of “templates”, convenient for reusing part of a program. We present the explanation formalism, then its ASP translation in DLV-Complex, and we conclude by a few reasonable expectations about the future ASP systems which could help a final user.

2 The causal explanation formalism

2.1 Preliminaries (propositional version, cf [2] for the full formalism).

We distinguish various types of statements:

C: A theory expressing causal statements. E.g. *On_alarm causes Heard_bell*.

O: *IS-A* links between items which can appear in a causal statement. E.g.,
Temperature_39 \rightarrow_{IS-A} *Fever_Temperature*,
Heard_soft_bell \rightarrow_{IS-A} *Heard_bell*.

W: A classical propositional theory expressing truths (incompatible facts, co-occurring facts, ...). E.g., *Heard_soft_bell* \rightarrow \neg *Heard_loud_bell*.

Propositional symbols denote states of affairs, which can be “facts” or “events” such as *Fever_Temperature* or *On_alarm*. The causal statements express causal relations between facts or events.

Some care is necessary when providing these causal and ontological atoms. If “*Flu causes Fever_Temperature*”, we conclude *Flu explains Temperature_39* from *Temperature_39* \rightarrow_{IS-A} *Fever_Temperature*, but we cannot state *Flu causes Temperature_39*: the causal information must be “on the right level”.

The formal system infers formulas denoting explanations from $C \cup O \cup W$. The *IS_A* atoms express knowledge necessary to infer explanations. In the following, α, β, \dots denote the propositional atoms and Φ, Ψ, \dots denote sets thereof.

Atoms

1. *Propositional atoms*: α, β, \dots
2. *Causal atoms*: α causes β .

3. *Ontological atoms*: $\alpha \rightarrow_{IS-A} \beta$. Reads: α is a β .
4. *Explanation atoms*: α explains β bec_poss Φ . Reads: α is an explanation for β because Φ is possible.

Formulas

1. *Propositional formulas*: Boolean combinations of propositional atoms.
2. *Causal formulas*: Boolean combinations of causal or propositional atoms.

The premises $C \cup O \cup W$ consist of propositional and causal formulas, and ontological *atoms* (no ontological formula), without explanation atom.

1. Properties of the causal operator

- (a) *Entailing [standard] implication*: If α causes β , then $\alpha \rightarrow \beta$.

2. Properties of the ontological operator

- (a) *Entailing implication*: If $\alpha \rightarrow_{IS-A} \beta$, then $\alpha \rightarrow \beta$.
- (b) *Transitivity*: If $a \rightarrow_{IS-A} b$ and $b \rightarrow_{IS-A} c$, then $a \rightarrow_{IS-A} c$.
- (c) *Reflexivity*: $c \rightarrow_{IS-A} c$. (unconventional, keeps the number of rules low).

2.2 The formal system

1. *Causal atoms entail implication*: $(\alpha \text{ causes } \beta) \rightarrow (\alpha \rightarrow \beta)$.

2. *Ontological atoms*

- (a) entail implication: If $\beta \rightarrow_{IS-A} \gamma$ then $\beta \rightarrow \gamma$.
- (b) transitivity: If $\alpha \rightarrow_{IS-A} \beta$ and $\beta \rightarrow_{IS-A} \gamma$ then $\alpha \rightarrow_{IS-A} \gamma$.
- (c) reflexivity: $\alpha \rightarrow_{IS-A} \alpha$

3. *Generating the explanation atoms*

- (a) *Initial case* If $\delta \rightarrow_{IS-A} \beta$, $\delta \rightarrow_{IS-A} \gamma$, and $W \not\models \neg(\alpha \wedge \delta)$,
then $(\alpha \text{ causes } \beta) \rightarrow \alpha \text{ explains } \gamma \text{ bec_poss } \{\alpha, \delta\}$.
- (b) *Transitivity (gathering the conditions)* If $W \not\models \neg \wedge(\Phi \cup \Psi)$,
then $(\alpha \text{ explains } \beta \text{ bec_poss } \Phi \wedge \beta \text{ explains } \gamma \text{ bec_poss } \Psi) \rightarrow$
 $\alpha \text{ explains } \gamma \text{ bec_poss } (\Phi \cup \Psi)$.
- (c) *Simplification of the set of conditions* If $W \models \wedge \Phi \rightarrow \bigvee_{i=1}^n \wedge \Phi_i$, then
 $\bigwedge_{i \in \{1, \dots, n\}} \alpha \text{ explains } \beta \text{ bec_poss } (\Phi_i \cup \Phi) \rightarrow \alpha \text{ explains } \beta \text{ bec_poss } \Phi$.

The elementary “initial case” applies (2c) upon (3a) where $\beta = \gamma = \delta$, together with a simplification (3c) since $\alpha \rightarrow \beta$ here, getting:

If α *causes* β and $W \not\models \neg\alpha$ then α *explains* β *bec_poss* $\{\alpha\}$.

These rules are intended as a compromise between expressive power, naturalness of description and relatively efficiency.

Transitivity of *explanations* occurs (gathering conditions). The simplification rule (3c) is powerful and costly, so the ASP translation implements the following weaker rule (also, it never removes $\{\alpha\}$ from Φ): [3c'] If $W \models \bigwedge \Phi - \{\varphi\} \rightarrow \bigwedge \Phi$, and α *explains* β *bec_poss* Φ then α *explains* β *bec_poss* $\Phi - \{\varphi\}$.

An atom α *explains* β *bec_poss* Φ is *optimal* if there is no explanation atom α *explains* β *bec_poss* Ψ where $W \models \bigwedge \Psi \rightarrow \bigwedge \Phi$ and not conversely. Keeping only these weakest sets of conditions is useful when the derivation is made only thanks to the part of W coming from Points 1 and 2a above. This keeps all the relevant explanation atoms and is easier to read.

2.3 A generic diagram

The following diagram summarizes many patterns of inferred explanations:

α <i>causes</i> β ,	α <i>causes</i> β_0 ,	β_2 <i>causes</i> γ ,	β_1 <i>causes</i> γ ,
β_3 <i>causes</i> ϵ ,	γ_1 <i>causes</i> δ ,	γ_3 <i>causes</i> δ ,	ϵ_3 <i>causes</i> γ_3 ;
$\beta \rightarrow_{IS-A} \beta_2$,	$\beta_1 \rightarrow_{IS-A} \beta$,	$\beta_3 \rightarrow_{IS-A} \beta_0$,	$\beta_3 \rightarrow_{IS-A} \beta_1$,
$\gamma_1 \rightarrow_{IS-A} \gamma$,	$\gamma_2 \rightarrow_{IS-A} \gamma$,	$\gamma_2 \rightarrow_{IS-A} \gamma_3$,	$\gamma_2 \rightarrow_{IS-A} \epsilon$,
$\epsilon_1 \rightarrow_{IS-A} \epsilon$,	$\epsilon_2 \rightarrow_{IS-A} \epsilon$,	$\epsilon_1 \rightarrow_{IS-A} \epsilon_3$,	$\epsilon_2 \rightarrow_{IS-A} \epsilon_3$.

This example shows various different “explaining paths” from a few given causal and ontological atoms. As a first “explaining path” from α to δ we get successively (path (1a): α *explains* β_2 *bec_poss* $\{\alpha\}$, α *explains* γ_1 *bec_poss* $\{\alpha, \gamma_1\}$, and α *explains* δ *bec_poss* $\{\alpha, \gamma_1\}$, giving α *explains* δ *bec_poss* $\{\alpha, \gamma_1\}$ (1a). The four optimal paths from α to δ are depicted, and non optimal paths (e.g. going through β_1) exist also.

3 An ASP translation of the formalism

3.1 Presentation

We describe a program in DLV-Complex. A first version [8] used pure DLV, and was much slower and harder to read. We have successfully tested an example with more than a hundred symbols and more than 10 different explanation atoms for some (I, J) (made from two copies of the example of the diagram, linked through a few more data). We have encountered a problem, not listed in the “three problems” evoked above. The full program, including “optimization of the result”, did not work on our computer. The simplification step and the verification step are clearly separated from the first generating step, thus we have split the program in three parts: The first one generates various explanation atoms (including all the optimal ones). The second program keeps only the optimal explanation atoms, in order to help reading the set of the solutions,

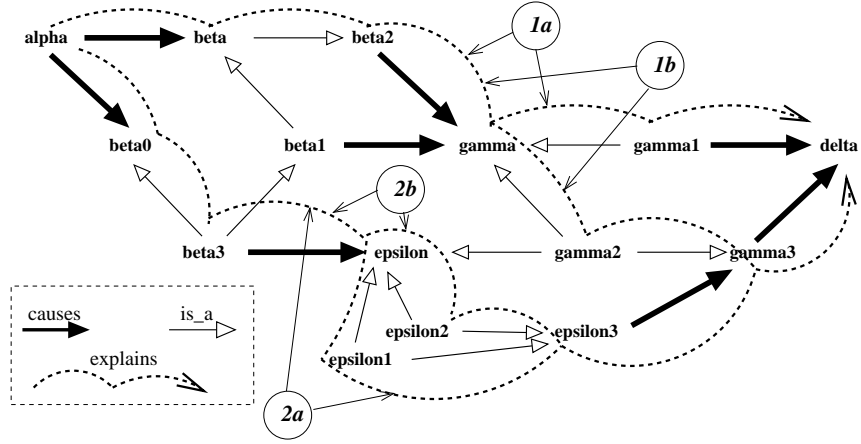


Figure 1: Four optimal explanation paths from “alpha” to “delta”

following Point 3c’. The third program checks whether the set of conditions is satisfied in the answer set considered. Then our “large example” works. Notice that it useful to enumerate the answer sets (as described in the simple and very interesting [11]) in order to help this splitting.

3.2 The generating part: getting the relevant explanation atoms

The “answer” of an ASP program is a set of *answer sets*, that is a set of concrete literals satisfying the rules (see e. g. [1] for exact definitions and [6] for the precise syntax of DLV (“:-” represents “ \leftarrow ” and “-” alone represents “standard negation” while “v” is the disjunction symbol in the head of the rules, and “!=” is “ \neq ”).

The user provides the following data:

- symbol(alpha) for each propositional symbol alpha.
- cause(alpha,beta) for each causal atom α *causes* β ,
- ont(alpha,beta) for each “is_a” atom $\alpha \rightarrow_{IS-A} \beta$ and
- true(alpha) for each other propositional atom α involved in formulas.

Causal and propositional formulas, such as $(\neg\epsilon1 \wedge \neg\epsilon2) \vee \neg\gamma1 \vee \neg\gamma2$ must be put in conjunctive normal form, in order to be entered as sets of clauses:

- { -true(epsilon1) v -true(gamma1) v -true(gamma2). ,
- true(epsilon2) v -true(gamma1) v -true(gamma2). }

The interesting result consists in the explanation predicates:

- ecSet(alpha,beta,{alpha,delta,gamma}) represents the explanation atom α *explains* β *bec_poss* { α, δ }.

Here come the first rules (cf 2b §2.2):

- ontt(l,j) :- ont(l,j). ont(t,l,k) :- ont(t,l,j), ont(j,k). ont(t,l,l) :- symbole(l).

We refer the reader to [9] for more details about the program. As an example of the advantage of using sets, let us give here the rule dealing with transitivity of explanations (cf 3b §2.2), `ecinit` referring to explanations not using transitivity rule. (`#insert(Set1,E2,Set)` means: $Set = Set1 \cup \{E2\}$):

```
ecSet(I,J,{I}) :- ecinit(I,J,I). ecSet(I,J,{I,E}) :- ecinit(I,J,E), not ecSet(I,J,{I}).
ecSet(I,J,Set) :- ecSet(I,K,Set1), not ecSet(I,J,Set1), ecinit(K,J,E2), E2 != K,
                  #insert(Set1,E2,Set).
ecSet(I,J,Set) :- ecSet(I,K,Set), ecinit(K,J,K).
```

3.3 Optimizing the explanation atoms

The “weak simplification” rule 3c’ §2.2) is used at this step, omitted here for lack of space. Moreover, if two sets of condition Φ , Ψ exist for some α *explains* β , and if $\Phi \models \Psi$ and not conversely, then only α *explains* β *bec_poss* Ψ is kept, the stronger set Φ being discarded. This avoids clearly unnecessary explanation atoms. This part (not logically necessary) is costly, but helps interpreting the result by a human reader.

3.4 Checking the set of conditions

Finally, the following program starts from the result of any of the last two preceding programs and checks, in each answer set, whether the set of conditions is satisfied or not. The result is given by `explVer(I,J,Set)`: *I explains J bec_poss Set* where *Set* is satisfiable in the answer set considered (“Ver” stands for “verified”).

```
explSuppr(I,J,Set) :- ecSetRes(I,J,Set), -true(E), #member(E,Set).
explVer(I,J,Set) :- ecSetRes(I,J,Set), not explSuppr(I,J,Set).
```

Only “individual” checking is made here, in accordance with the requirement that the computational properties remain manageable.

With the whole chain (§3.2, 3.3, and 3.4), modifying a rule of the formalism can be done easily. The gain of using DLV-Complex instead of pure DLV (or gringo/claspD) is significant and worth mentioning.

3.5 Conclusion and future work

We have shown how the recent versions of running ASP systems allow easy translation of logical formalisms. The example of the explanation formalism shows that such a translation can already be useful for testing new theories. In a near future, cases from the “real world” should be manageable and the end user should be able to use ASP for a great variety of diagnostic problems.

Here are two considerations about what could be hoped for future ASP systems in order to deal easily with this kind of problem. Since ASP systems are regularly evolving, we can hope that a near future the annoying trick consisting in launching the programs one after the other, and not in a single launch, should become unnecessary. It seems easy to detect that some predicates can safely be computed first, before launching the subsequent computation. In our example, computing `ecSet` first, then `ecSetRes` and finally `ecSetVer` is possible, and such

one way dependencies could be detected. The great difference in practice between launching the three programs together, and launching them one after the other, shows that such improvement could have spectacular consequences.

Also, efficient “enumerating meta-predicates” would be useful (even if logically useless). A last interesting improvement would concern the possibility of implementing “enumerating answer sets”, as described in e.g. [11], then complex comparisons between answer sets could be made.

For what concerns our own work, the important things to do are to apply the formalism to real situations, and, to this respect, firstly to significantly extend our notion of “ontology” towards a real one.

References

- [1] Chitta Baral, Knowledge representation, reasoning and declarative problem solving, Cambridge University Press, 2003.
- [2] Ph.Besnard, M-O Cordier and Y. Moinard, Ontology-based inference for causal explanation, *Integrated Computer-Aided Engineering J.*, IOS, 15(4): 351–367, 2008.
- [3] F. Calimeri and G. Ianni, Template programs for Disjunctive Logic Programming: An operational semantics, *AI Communications*, 19(3): 193–206, 2006, IOS Press
- [4] Giunchiglia E., Lee J., Lifschitz V., McCain N., Turner H. Nonmonotonic Causal Theories. *Artificial Intelligence* 153(1–2):49–104, 2004.
- [5] Halpern J. and Pearl J. Causes and Explanations: A Structural-Model Approach - Part II: Explanations. In *IJCAI-01*, pp. 27-34. Morgan Kaufmann, 2001.
- [6] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV System for Knowledge Representation and Reasoning. *ACM Trans. on Computational Logic (TOCL)*, 7(3):499–562, 2006.
- [7] Mellor D. H. *The Facts of Causation*. Routledge, 1995.
- [8] Yves Moinard An Experience of Using ASP for Toy Examples, ASP’07, in *Advances in Theory and Implementation*, Fac. de Ciencias, Univ. do Porto, pp. 133–147, 2007.
- [9] Yves Moinard Using ASP with recent extensions for causal explanations, ASPOCP Workshop, associated with ICLP, 2010
- [10] Shafer G. Causal Logic. In H. Prade (ed), *ECAI-98*, pp. 711-720, 1998.
- [11] L. Tari and Ch. Baral and S. Anwar, A Language for Modular Answer Set Programming: ASP 05, CEUR-WS.org publ., CEUR Workshop Proc., vol. 142, 2005.