Knowledge Representation and Reasoning in NMR and Extended LP

Jürgen Dix

University of Koblenz-Landau
Dept. of Computer Science

and

Technical University of Vienna
Institute for Knowledge-Based Systems

IV. Definite Logic Programs

- Top-Down
- Bottom-Up
- Herbrand Models and underlying Language
- Why going beyond Definite Programs?
- What is a Semantics?

Definition 0.3 (Definite Logic Program)
A definite *logic program consists of a finite number of* rules *of the form*

$$\boxed{A \leftarrow B_1, \ldots, B_m},$$

where $A, B_1, \ldots, B_m$ *are* $\boxed{\textit{positive atoms}}$ *(containing possibly free variables). We call* A *the* head *of the rule and* $B_1, \ldots, B_m$ *its* body.

Definition 0.4 (Query)
Given a definite program we usually have a definite query in mind that we want to be solved. A definite query Q is a conjunction of positive atoms $C_1 \wedge \ldots \wedge C_l$ which we denote by

$$?\text{-}\ \ C_1, \ldots, C_l.$$

These C_i may also contain variables. Asking a query Q to a program P means asking for all possible substitutions Θ of the variables in Q such that $Q\Theta$ follows from P. Often, Θ is also called an answer to Q. Note that $Q\Theta$ may still contain free variables.

Given two atoms A and B with free variables we can ask if we can compute two substitutions Θ_1, Θ_2 for the variables s.t.

$$A\Theta_1 \text{ is identical to } B\Theta_2.$$

In fact, this problem is decidable and if the two atoms are *unifiable* we can indeed compute a *most general unifier*, called mgU (see (Lloyd 1987)). An mgU $\boxed{\Theta}$ for A and B is a substitution defined on all variables ocurring in both A and B such that

$$A\Theta \text{ is identical to } B\Theta.$$

How are our programs related to classical predicate logic? Can we simply translate rules into classical formulae via

$$\boxed{B_1 \wedge \ldots \wedge B_m \supset A}?$$

Principle 0.5 (Orientation)
If a ground atom A does not unify with some head of a program rule of P, then this atom is considered to be false. In this case we say that $\boxed{not\ A}$ *is derivable from P to distinguish it from classical* $\boxed{\neg A}$.

1 Top-Down

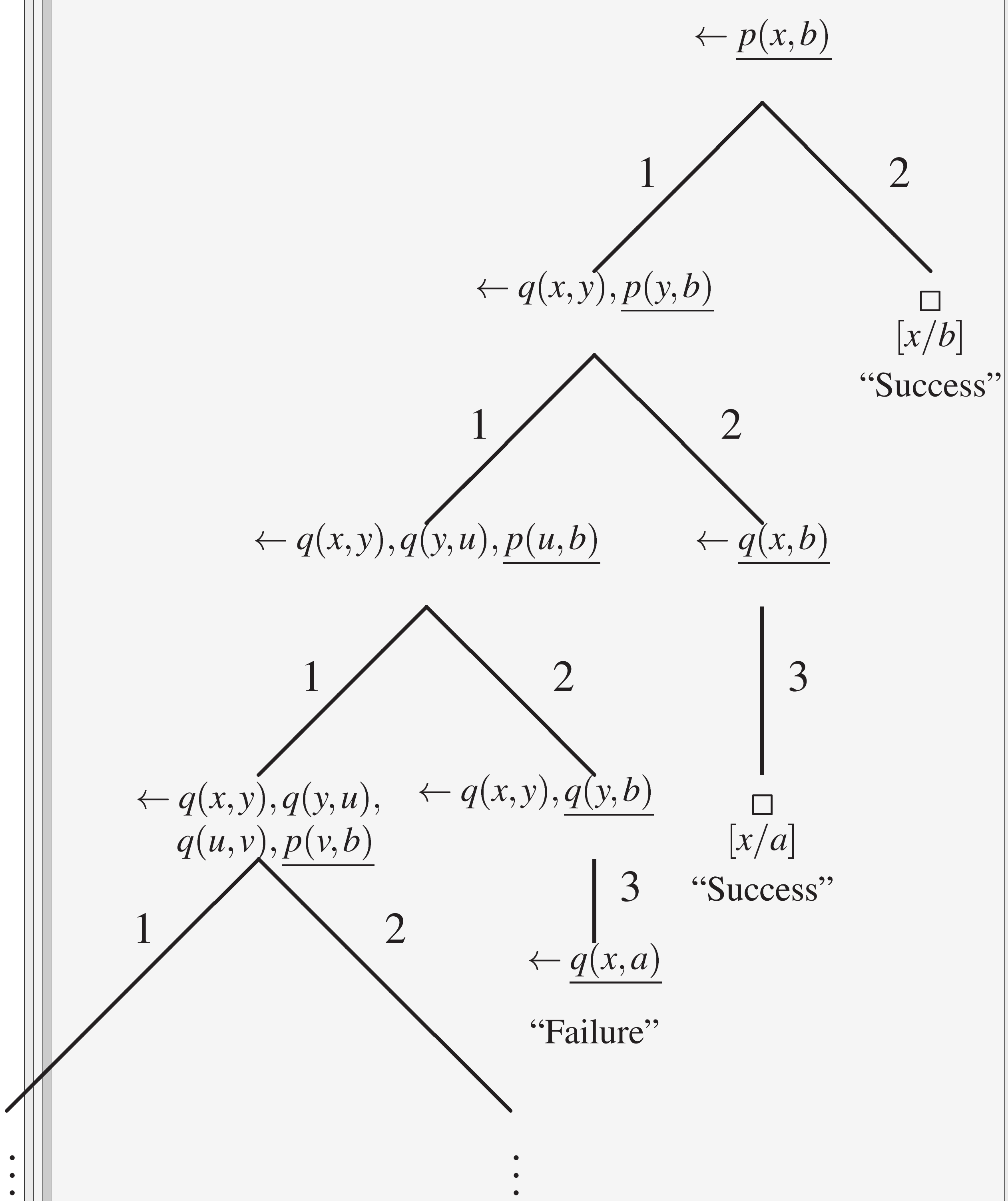
Example 1.1 (SLD-Resolution)

Let the program P_{SLD} consist of the following three clauses

$$
\begin{array}{llll}
(1) & p(x,z) & \leftarrow & q(x,y), p(y,z) \\
(2) & p(x,x) & & \\
(3) & q(a,b) & &
\end{array}
$$

The query Q we are interested in is given by $p(x,b)$. I.e. we are looking for all

substitutions Θ for x such that $p(x,b)\Theta$ follows from P.


← p(x,b)
1
2
← q(x,y), p(y,b)
□
[x/b]
“Success”
1
2
← q(x,y), q(y,u), p(u,b)
← q(x,b)
1
2
3
← q(x,y), q(y,u),
q(u,v), p(v,b)
← q(x,y), q(y,b)
□
[x/a]
“Success”
3
1
2
← q(x,a)
“Failure”

Obviously, there are three different sorts of branches, namely

1. *infinite* branches,
2. branches that *end up with the empty clause* “Success”, and
3. branches that *end in a deadlock* (*“Failure”*): no applicable rule is left.

Theorem 1.2 (Soundness and Completeness of SLD)
The following properties are equivalent:

- $P \models \forall\, Q\Theta$, *i.e.* $\forall\, Q\Theta$ *is true in all models of* P,
- $M_P \models \forall\, Q\Theta$,
- *SLD computes an answer* τ *that subsumes* Θ *wrt* Q
  (i.e. $\exists\sigma : Q\tau\sigma = Q\Theta$.*)*

Note that not any correct answer is computed,
only the most general one is.

The main feature of SLD-Resolution is its *Goal-Orientedness*.

2 Bottom-Up

Example 2.1 (T_P)
Given a definite program P let $\boxed{T_P}$: $2^{B_P} \longmapsto 2^{B_P}$; $I \longmapsto T_P(I)$

$$\boxed{T_P}(I) := \{A \in B_P : \begin{array}{l}\textit{there is an instantiation of a rule in } P \\ \textit{s.t. } A \textit{ is the head of this rule and all} \\ \textit{body-atoms are contained in } I \,\}\end{array}$$

It turns out that $\boxed{T_P}$ is monotone and continuous: so that

Theorem 2.2 (T_P **and** M_P)
$M_P = \boxed{T_P}\uparrow^{\omega} = lfp(\boxed{T_P})$.

This approach is especially important in Databases, where the underlying language does not contain function symbols (DATALOG). In addition, rules of the form

$$\boxed{p \leftarrow p}$$

do not make any problems.

Principle 2.3 (Elimination of Tautologies)
Suppose a program P has a rule which contains the same atom in its body as well as in its head (i.e. the head consists of exactly this atom). Then we can eliminate this rule without changing the semantics.

Two main Shortcomings of Bottom-Up:

- no Goal-orientedness,
- Many facts are recomputed again and again.

3 Herbrand Models

Why are we considering all models? Doesn't it make sense to look only at models generated by the underlying language?

Definition 3.1 (UNA and DCA)

Let a language $\mathcal{L}$ be given. We understand by the Unique Names Assumption *the restriction to those models I, where* *syntactically different ground $\mathcal{L}$-terms t_1, t_2* *are interpreted as* *nonidentical elements*: *t_1^I is not identical to t_2^I.*

By the Domain Closure Assumption *we mean the restriction to those models I where*

for any element a in I there is a $\mathcal{L}$-term t that represents this element: $a = t^I$.

Herbrand models I are models satisfying UNA, DCA and: a term "$f(t_1, \ldots, t_n)$" is interpreted in I as "$\boxed{f}(\boxed{t_1^I}, \ldots, \boxed{t_n^I})$".

For universal theories T and existential formulae ϕ the following holds

$$\boxed{\mathrm{MOD}(T) \models \phi \quad \text{iff} \quad \mathrm{Herb}_{\mathcal{L}}\text{-}\mathrm{MOD}(T) \models \phi.}$$

In our particular case, where T is a definite program P, we can even replace $\mathrm{Herb}_{\mathcal{L}}\text{-}\mathrm{MOD}(T)$ in the above equation by the single model M_P.

4 Why going beyond?

In KR we do not only want to formulate negative queries, we also want to express *default-statements* of the form

Normally, unless something abnormal holds: ψ implies ϕ.

How can we formulate such a statement as a logic program? The most natural way is to use negation "*not* "

$$\phi \leftarrow \psi, \boxed{\mathit{not}}\ ab$$

where *ab* stands for *abnormality*. Obviously, this forces us to extend definite programs by negative atoms.

Example 4.1 (Inheritance Hierachies)
Suppose we know that birds typically fly and penguins are non-flying birds. We also know that Tweety is a bird. Now an agent is hired to build a cage for Tweety. Should the agent put a roof on the cage? After all it could be still the case that Tweety is a penguin and therefore can not fly, in which case we would not like to pay for the unneccessary roof. But under normal conditions, it should be obvious that one should conclude that Tweety is flying.

A natural axiomatization is given as follows:

$$
\begin{array}{llll}
P_{Inheritance}: & flies(x) & \leftarrow & bird(x), \quad \boxed{not}\ ab(r_1, x) \\
& bird(x) & \leftarrow & penguin(x) \\
& ab(r_1, x) & \leftarrow & penguin(x) \\
& make\_top(x) & \leftarrow & flies(x)
\end{array}
$$

together with some particular facts, like $bird(Tweety)$ and $penguin(Sam)$.

For the query "$make\_top(Tweety)$" we expect the answer "yes" while for "$make\_top(Sam)$" we expect the answer "no".

Example 4.2 (Temporal Projection: Yale-Shooting)
We distinguish between three sorts of variables:

- *situation variables*: $S, S', \ldots,$
- *fluent variables*: $F, F', \ldots,$
- *action variables*: $A, A', \ldots.$

The initial situation is denoted by the constant s_0, and the two-ary function symbol $res(A,S)$ denotes the situation that is reached when in situation S the action A has been performed. The relation symbol $holds(F,S)$ formalizes that the fluent F is true in situation S.

The initial situation is denoted by the constant s_0, and the two-ary function symbol $res(A,S)$ denotes the situation that is reached when in situation S the action A has been performed. The relation symbol $holds(F,S)$ formalizes that the fluent F is true in situation S.

For the YSP there are three actions $\boxed{wait}$, $\boxed{load}$ and $\boxed{shoot}$) and two fluents $alive$ and $loaded$). Initially a turkey called $Fred$ is alive. We then load a gun, wait and shoot. The effect should be that Fred is dead after this sequence of actions. The common-sense argument from which this should follow is the

Law of Inertia: Things normally tend to stay the same.

Using our intuition from the last example, a natural formalization is given as follows:

$$
\begin{array}{lll}
P_{YSP}: holds(F, res(A,S)) & \leftarrow & holds(F,S),\ \boxed{not}\ ab(r_1, A, F, S) \\
holds(loaded, res(load, S)) & \leftarrow & \\
ab(r_1, shoot, alive, S) & \leftarrow & holds(loaded, S) \\
ab(r_1, shoot, loaded, S) & \leftarrow & holds(loaded, S) \\
holds(alive, s_0) & \leftarrow &
\end{array}
$$

Such a straightforward formalization leads in most versions of classical nonmonotonic logic to the unexpected result, that Fred is not neccesarily dead. We expect to derive

$$holds(alive, res(load, s_0))$$

$$\boxed{not}\ holds(alive\ , res(shoot, res(wait, res(load, s_0))))$$

Up to now we only have stated some very “natural” axiomatizations of given knowledge. We have motivated that something like default-negation “*not* ” should be added to definite programs in order to do so and we have explicitly stated the answers to particular queries. What is still missing are solutions to the following very important problems

- *How should an appropriate query answering mechanism handling default-negation “not ” look like?*
- *What is the formal semantics that such a procedural mechanism should be checked against?*

5 What is a Semantics

In the last subsections we have introduced two principles (*Orientation* and *Elimination of Tautologies*) and used the term *semantics of a program* in a loose, imprecise way.

Definition 5.1 (SEM)
A semantics SEM is a mapping from the class of all programs into the powerset of the set of all 3-valued structures. SEM assigns to every program P a set of 3-valued models of P:

$$\boxed{SEM(P) \subseteq MOD_{3-val}{}^{\mathcal{L}_P}(P).}$$

Formally, we can associate to any semantics SEM in the sense of Definition 5.1 two entailment relations

sceptical: $\mathrm{SEM}^{scept}(P)$ is the set of all atoms or default atoms that are true in *all models* of $\mathrm{SEM}(P)$.

credulous: $\mathrm{SEM}^{cred}(P)$ is the set of all atoms or default atoms that are true in *at least one model* of $\mathrm{SEM}(P)$.

When there is only one model, we will omit the outer brackets and write (instead of $\mathrm{SEM}(P) = \{M\}$)

$$\mathrm{SEM}(P) = M.$$

We will also slightly abuse notation and write $l \in \mathrm{SEM}(P)$ as an abbreviation for $l \in M$ for all $M \in \mathrm{SEM}(P)$.

References

Berman, K. A., J. S. Schlipf, and J. V. Franco (1995, June). Computing the Well-Founded Semantics Faster. In A. Nerode, W. Marek, and M. Truszczyński (Eds.), *Logic Programming and Non-Monotonic Reasoning, Proceedings of the Third International Conference*, LNCS 928, Berlin, pp. 113–126. Springer.

Brass, S. and J. Dix (1995, September). D-WFS: A Confluent calculus and an Equivalent Characterization. Technical Report TR 12/95, University of Koblenz, Department of Computer Science, Rheinau 1.

Brass, S. and J. Dix (1996). Characterizing D-WFS: Confluence and Iterated GCWA. In L. P. J.J. Alferes and E. Orlowska (Eds.), *Logics in Artificial Intelligence (JELIA '96)*, LNCS 1126, pp. 268–283. Springer. (Extended version will appear in the *Journal of Automated Reasoning* in 1998.).

Dix, J. and P. Schmitt (1993). Nichtmonotones Schließen: Wieviel Nichtmonotonie ist nötig? *Kognitionswissenschaft 3*, 53–69.

Dix, J. and F. Stolzenburg (1998). A Framework to incorporate Nonmonotonic Reasoning into Constraint Logic Programming. *Journal of Logic Program-*

ming 36(1,2,3), 5—37. Special Issue on *Constraint Logic Programming*, Guest Editors: Kim Marriott and Peter Stuckey.

Eiter, T. and G. Gottlob (1993). Propositional Circumscription and Extended Closed World Reasoning are Π_2^P-complete. *Theoretical Computer Science 144*(2), 231–245, Addendum: vol. 118, p. 315, 1993.

Eiter, T. and G. Gottlob (1995). On the Computational Cost of Disjunctive Logic Programming: Propositional Case. *Annals of Mathematics and Artificial Intelligence 15*(3/4), 289–323.

Gottlob, G. (1992). Complexity results for nonmonotonic logics. *Journal of Logic and Computation 2*(3), 397–425.

Lehmann, D. and M. Magidor (1992). What does a conditional knowledge base entail? *Artificial Intelligence 55*, 1–60.

Lifschitz, V. (1989, January). Benchmark Problems for Formal NMR, Version 2.00. In Reinfrank, de Kleer, Ginsberg, and Sandewall (Eds.), *Non-Monotonic Reasoning*, LNAI 346, Berlin, pp. 202–219. Springer.

Lloyd, J. W. (1987). *Foundations of Logic Programming*. Berlin: Springer. 2nd edition.

McCarthy, J. (1986). Applications of Circumscription to formalizing Common-Sense Reasoning. *Artificial Intelligence 28*, 89–116.

Niemelä, I. and P. Simons (1996, September). Efficient Implementation of the Well-founded and Stable Model Semantics. In M. Maher (Ed.), *Proceedings of the Joint International Conference and Symposium on Logic Programming*, Bonn, Germany, pp. 289–303. The MIT Press.

Reiter, R. (1980). A Logic for Default-Reasoning. *Artificial Intelligence 13*, 81–132.

Schlipf, J. (1986, June). How uncomputable is general circumscription? In *Proceedings of the Symposium on Logic in Computer Science*, Cambridge, USA, pp. 92–95. IEEE Computer Society Press.

Stillman, J. (1992, July). The complexity of propositional default logics. In *Proceedings of the 10th National Conference on Artificial Intelligence*, San Jose, California, USA, pp. 794–800. The MIT Press.

Tarjan, R. (1972). Depth first search and linear graph algorithms. *SIAM Journal on Computing 1*, 146–160.

Van Gelder, A. (1989, March). The Alternating Fixpoint of Logic Programs with Negation. In *Proc. of the Eight ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Philadelphia, Pennsylvania*, pp. 1–10. ACM Press.

Van Gelder, A., K. A. Ross, and J. S. Schlipf (1988). Unfounded Sets and well-founded Semantics for general logic Programs. In *Proceedings 7th Symposion on Principles of Database Systems*, pp. 221–230.