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

V. Adding Default Negation

- Negation-as-Finite-Failure
- Negation-as-Failure
- Wellfounded Semantics: WFS
- Stable Semantics: STABLE

We can distinguish between two quite different approaches:

LP-Approach: This is the approach taken mainly in the Logic Programming community. There one tried to stick as close as possible to SLD-Resolution and treat negation as "Finite-Failure". This resulted in SLDNF-Resolution.

NML-Approach: This is the approach suggested by non-monotonic reasoning people. Here the main question is *"What is the right semantics?"* I.e. we are looking first for a semantics that correctly fits to our intuitions and treats the various KR-Tasks in the right (or appropriate) way.

1 Negation-as-Finite-Failure

How should we handle default-atoms?

- If we reach "*not A*" as a subgoal, we keep the current SLD-tree in mind and start a new SLD-tree by trying to solve "*A*".
- If this succeeds, then we falsified "*not A*", the current branch is failing and we have to backtrack and consider a different subquery.
- But it can also happen that the SLD-tree for "*A*" is *finite with only failing branches*. Then we say that *A finitely fails*, we turn back to our original SLD-tree, consider the subgoal "*not A*" as successfully solved and go on with the next subgoal in the current list.

Principle 1.1 (A "Naive" SLDNF-Resolution)

If in the construction of an SLDNF-tree a default-atom not L_{ij} is selected in the list $\mathcal{L}_i = \{L_{i1}, L_{i2}, \ldots\}$, then we try to prove L_{ij}.

If this fails finitely *(it fails because the generated subtree is finite and failing), then we take not L_{ij} as proved and we go on to prove $L_{i(j+1)}$.*

If L_{ij} succeeds, then not L_{ij} fails and we have to backtrack to the list $\mathcal{L}_{i-1}$ of preliminary subgoals (the next rule is applied: "backtracking").

SLDNF handles properly our examples 4.1 and 4.2.

Up to now it seems that SLDNF-resolution solves all our problems. It handels our examples correctly, and is defined by a procedural calculus strongly related to SLD. There are two main problems with SLDNF:

- SLDNF can not handle free variables in negative subgoals,
- SLDNF is still too weak for Knowledge Representation.

The latter problem is the most important one.

Floundering

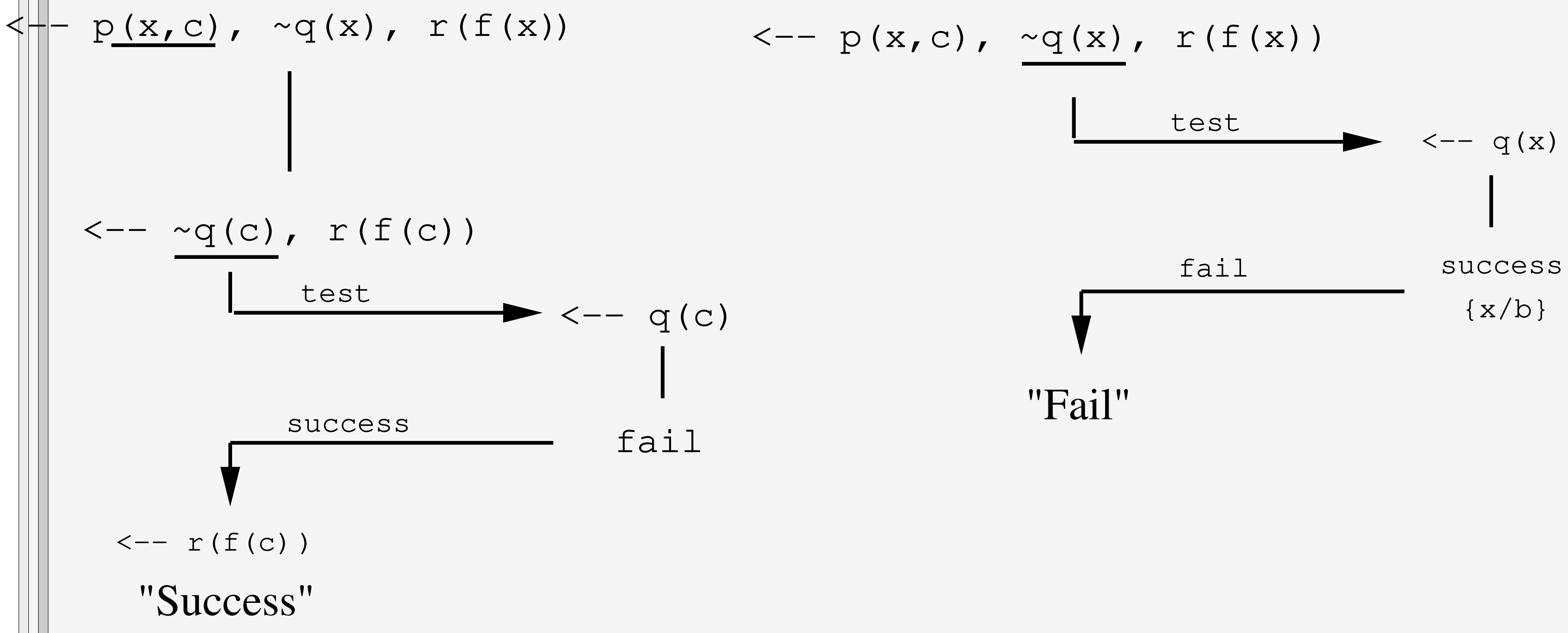

Figure 3: The Floundering-Problem

2 Negation-as-Failure

SLDNF answers quite easily our requirements of a semantics SEM (stated explicitly in Definition 5.1).

Principle 2.1 (Reduction)
Suppose we are given a program P with possibly default-atoms in its body. If a ground atom A does not unify with any head of the rules of P, then we can delete in every rule any occurrence of "not A" without changing the semantics.

Dually, if there is an instance of a rule of the form "$B \leftarrow$ " then we can delete all rules that contain "not B" in their bodies.

When we consider rules of the form "$p \leftarrow p$", then SLD resolution gets into an infinite loop and no answer to the query *?-* p can be obtained. This has often the effect that when we enter into negation-as-failure mode, the SLD-tree to be constructed is not finite, although he is not successful and therefore should be considered as failed.

Example 2.2 (The Transitive Closure)
Assume we are given a graph consisting of nodes and edges between some of them. We want to know which nodes are reachable from a given one. A natural formalization of the property "reachable" would be

$$reachable(x) \leftarrow edge(x,y), reachable(y).$$

What happens if we are given the following facts

$$edge(a,b),\ edge(b,a),\ edge(c,d)$$

and $reachable(c)$? Of course, we expect that neither a nor b are reachable because there is no path from c to either a or b.

But SLDNF-Resolution does not derive "not $reachable(a)$"!

Partial Evaluation

The query “*not* $reachable(a)$” leads to the rule “$reachable(a) \leftarrow edge(a,b), reachable(b)$” and “$reachable(b)$” leads to “$reachable(b) \leftarrow edge(b,a), reachable(a)$”. Both rules are definitions for $reachable(a)$ and $reachable(b)$. So we can replace the body atoms of $reachable$ by their definitions:

$$\boxed{reachable(a)} \leftarrow edge(a,b), \boxed{edge(b,a), reachable(a)}$$
$$\boxed{reachable(b)} \leftarrow edge(b,a), \boxed{edge(a,b), reachable(b)}$$

that can both be eliminated by applying Principle 2.3.

So we end up with a program that does neither contain $reachable(a)$ nor $reachable(b)$ in one of the heads. Therefore, according to Principle 0.5 both atoms should be considered false.

Definition 2.3 (GPPE)
A semantics SEM satisfies GPPE if the following transformation does not change SEM:
Replace a rule $A \leftarrow \mathcal{B}^+ \wedge not\ \mathcal{B}^-$ where $\mathcal{B}^+$ contains an atom B by the rules

$$A \leftarrow \left(\mathcal{B}^+ \setminus \{B\}\right) \cup \mathcal{B}_i^+ \wedge not\ \left(\mathcal{B}^- \cup \mathcal{B}_i^-\right)(i = 1, \dots, n)$$

where $B \leftarrow \mathcal{B}_i^+ \wedge not\ \mathcal{B}_i^-$ $(i = 1, \dots, n)$ are all rules with head B.

$$\begin{array}{lcl} B & \leftarrow & not\ E \\ B & \leftarrow & D, not\ C' \\ A & \leftarrow & B, not\ C \end{array} \quad \mapsto \quad \boxed{\begin{array}{lcl} A & \leftarrow & D, not\ C', not\ C \\ A & \leftarrow & not\ E, not\ C \end{array}}$$

Note that any semantics SEM satsfying GPPE and Elimination of Tautologies can be seen as extending SLD by doing some *Loop-checking*. We will call such semantics *NMR-semantics* in order to distinguish them from the classical *LP-semantics* which are based on SLDNF or variants of Clark's completion $comp(P)$:

- *NMR-Semantics* = *SLDNF* + *Loop-check.*

Example 2.4 (COMP vs. NMR)

$$P_{NMR}: \begin{array}{lcl} p & \leftarrow & p \\ q & \leftarrow & not\ p \end{array} \qquad P'_{NMR}: \begin{array}{lcl} p & \leftarrow & p \\ q & \leftarrow & not\ p \\ r & \leftarrow & not\ r \end{array}$$

$$comp(P_{NMR}): \begin{array}{lcl} p & \equiv & p \\ q & \equiv & \neg p \end{array} \qquad comp(P'_{NMR}): \begin{array}{lcl} p & \equiv & p \\ q & \equiv & \neg p \\ r & \equiv & \neg r \end{array}$$

?-q: *No (COMP).* *Yes (NMR).*

?-p: *Yes (COMP).* *No (NMR).*

The last principle in this section is related to *Subsumption*: we can get rid of non-minimal rules by simply deleting them.

Principle 2.5 ($\boxed{\textbf{Subsumption}}$)

In a program P we can delete a rule $A \leftarrow \mathcal{B}^+ \wedge not\ \mathcal{B}^-$ whenever there is another rule $A \leftarrow \mathcal{B}'^+ \wedge not\ \mathcal{B}'^-$ with

$$\mathcal{B}'^+ \subseteq \mathcal{B}^+ \text{ and } \mathcal{B}'^- \subseteq \mathcal{B}^- .$$

$$\boxed{\begin{array}{lcl} A & \leftarrow & D, not\ E, not\ F \\ A & \leftarrow & D, not\ F \end{array}} \quad \mapsto \quad \boxed{\begin{array}{lcl} A & \leftarrow & D, not\ F \end{array}}$$

3 Wellfounded Semantics: WFS

We call a semantics SEM_1 weaker than SEM_2, if for all programs P and all atoms or default-atoms l the following holds: $\mathrm{SEM}_1(P) \models l$ implies $\mathrm{SEM}_2(P) \models l$.

I.e. all (default-) atoms derivable from SEM_1 with respect to P are also derivable from SEM_2.

Theorem 3.1 (WFS)

There exists the weakest semantics satisfying our four principles Elimination of Tautologies, Subsumption, Reduction, and GPPE. This semantics is called wellfounded semantics WFS.

Theorem 3.2 (Confluent Calculus for WFS)
The calculus consisting of these four transformations is confluent, i.e. whenever we arrive at an irreducible program, it is uniquely determined. The order of the transformations does not matter.

For finite propositional programs, it is also $\boxed{\textit{strongly terminating}}$ *for* $\boxed{\textit{fair sequences}}$*: any program P is therefore associated a unique normalform* $\boxed{res(P)}$*. The wellfounded semantics of P can be read off from* $\boxed{res(P)}$ *as follows*

$$
\begin{aligned}
WFS(P) = \quad & \{A : A \leftarrow \ \in \boxed{res(P)}\} \\
\cup \quad & \{not\, A : A \text{ is in no head of } \boxed{res(P)}\}
\end{aligned}
$$

Example 3.3 (Van Gelder's Example)
Assume we are describing a two-players game like checkers. The two players alternately move a stone on a board. The moving player wins when his opponent has no more move to make. We can formalize that by

- $\boxed{wins(x)} \leftarrow move\_from\_to(x,y), \boxed{not\ \ wins(y)}$

meaning that

- *the situation x is won (for the moving player A), if he can lead over*[a] *to a situation y that can never be won for B.*

---

[a]With the help of a regular move, given by the relation $move\_from\_to(,)$.

Assume we also have the facts

$$move\_from\_to(a,b), move\_from\_to(b,a), move\_from\_to(b,c).$$

Our query to this program P_{game} is *?- $wins(b)$*. Here we have no problems with floundering, but using SLDNF we get an infinite sequence of oscillating SLD-trees (none of which finitely fails).

$$WFS(P_{game}) = \{not\ wins(c), wins(b), not\ wins(a)\}$$

4 Stable Semantics: STABLE

The stable semantics associates to any program P a set of 2-valued models. STABLE also satisfies

Principle 4.1 (Elimination of Contradictions)
Suppose a program P has a rule which contains the same atom A and not A in its body. Then we can eliminate this rule without changing the semantics.

$$\text{Contradiction:} \quad A \vee B \quad \leftarrow \quad C, D, \mathit{not}\ C$$

Theorem 4.2 (STABLE)
There exists the weakest semantics satisfying our five principles Elimination of Tautologies, Subsumption, Reduction, GPPE and Elimination of Contradictions.

The underlying idea of STABLE is that any atom in an intended model should have a definite reason to be true or false. Gelfond-Lifschitz transformation: for a program P and a model $\boxed{N} \subseteq B_P$ we define

$$P^{\boxed{N}} := \{rule^{\boxed{N}} : \; rule \in P\}$$

where $rule := A \leftarrow B_1, \ldots, B_n, not\, C_1, \ldots, not\, C_m$ is transformed as follows

$$(rule)^{\boxed{N}} := \begin{cases} A \leftarrow B_1, \ldots, B_n, & \text{if } \forall j : C_j \notin \boxed{N}, \\ \mathbf{t}, & \text{otherwise.} \end{cases}$$

Note that P^N is always a *definite* program. We can therefore compute its least Herbrand model M_{P^N} and check whether it coincides with the model N with which we started:

Definition 4.3 (STABLE)

N *is called a* stable *model*[a] *of* P if and only if $M_{P^N} = N$.

[a] Note that we only consider Herbrand models.

What is the relationship between STABLE and WFS? We have seen that they are based on rather identical principles.

- Stable models N extend WFS: $l \in \mathrm{WFS}(P)$ implies $N \models l$.
- If WFS(P) is two-valued, then WFS(P) is the unique stable model.

For Example 3.3 we have two stable models: $\{wins(a), wins(c)\}$ and $\{wins(b), wins(c)\}$ and therefore

$$\mathrm{WFS}(P) = \{wins(c), not\ wins(d)\} = \bigcap_{\mathcal{N} \text{ a stable model of } P} \mathcal{N}.$$

Example 4.4 (Reasoning by cases)

$$
\begin{array}{llll}
P_{splitting}: & a & \leftarrow & not\ b \\
 & b & \leftarrow & not\ a \\
 & p & \leftarrow & a \\
 & p & \leftarrow & b
\end{array}
$$

Although neither a, nor b can be derived in any semantics based on two-valued models (as STABLE for example), the disjunction $a \vee b$, thus also p, is true. In this way the example is handled by the completion semantics, too. WFS(P), however, is empty; if the WFS cannot decide between a or not a, then a is undefined.

The main differences between STABLE and WFS are

- STABLE is not always consistent:

$$\text{STABLE}(p \leftarrow \textit{not } p) = \emptyset.$$

- STABLE does not allow for a goal-oriented implementation.

Example 4.5 (STABLE is not Goal-Oriented)

$$P_{rel(a)}: \begin{array}{lcl} a & \leftarrow & not\ b \\ b & \leftarrow & not\ a \end{array} \qquad P: \begin{array}{lcl} a & \leftarrow & not\ b \\ b & \leftarrow & not\ a \\ p & \leftarrow & not\ p \\ p & \leftarrow & a \end{array}$$

$P_{rel(a)}$ is the subprogram of P that consists of all rules that are relevant to answer the query ?- a. It has two stable models $\{a\}$ and $\{b\}$ — a is not true in all of them. But the program P has the unique stable model $\{p,a\}$, so a is true in all stable models of P.

- Can we make STABLE consistent by simply building consistency in?
  Yes!

- Can we make STABLE relevant by simply building Relevance in?
  No!

Example 4.6 (Modified STABLE and Relevance)

P :

$$\begin{aligned} a &\leftarrow not\ b \\ b &\leftarrow not\ a \\ x &\leftarrow a, not\ x \\ y &\leftarrow b, not\ y \\ w &\leftarrow not\ x, not\ y \end{aligned}$$

$P_{rel(x)}$:

$$\begin{aligned} a &\leftarrow not\ b \\ b &\leftarrow not\ a \\ x &\leftarrow a, not\ x \end{aligned}$$

$P_{rel(y)}$:

$$\begin{aligned} a &\leftarrow not\ b \\ b &\leftarrow not\ a \\ y &\leftarrow b, not\ y \end{aligned}$$

Both not x and not y are true in P (exactly one stable model exist for $P_{rel(x)}$ and $P_{rel(y)}$). Therefore w should also be true, but it is not, because P does not have stable models.

VI. Adding Disjunction

- GCWA
- D-WFS
- DSTABLE
- Complexity and Expressibility

In Knowledge Representation it often occurs that we know $A \vee B$ without being sure which of these propositions hold. We can have information that A implies D and B implies D from which we would like to derive that D holds for sure.

Definition 0.7 (CWA)

$$\mathrm{CWA}(\mathrm{DB}) = \mathrm{DB} \cup \{\neg P(t) : \ \mathrm{DB} \not\models P(t)\} ,$$

where $P(t)$ is a ground predicate instance.

1 GCWA

GCWA is defined for positive disjunctive programs

$$\boxed{A_1 \vee \ldots \vee A_n \leftarrow B_1, \ldots, B_m}$$

by declaring all the minimal models to be the intended ones:

Definition 1.1 (GCWA)
The generalized closed world assumption GCWA of P is the semantics given by the set of all minimal Herbrand models:

$$GCWA(P) := \mathit{Min\text{-}MOD}(P)$$

Originally, Minker denoted by GCWA(P) a certain set of negative atoms, but we prefer here to denote by GCWA a semantics in the sense of Definition 5.1.

GCWA is very important because

1. it plays the same role for positive disjunctive programs as the least Herbrand model M_P does for definite programs.

2. most semantics SEM defined for arbitrary disjunctive programs (i. e. with default-negation) can be characterized by reducing them to positive programs and then applying recursively GCWA.

3. in fact, an appropriate procedure iterating GCWA can "implement" such semantics SEM.

Theorem 1.2 (Characterization of GCWA)
Let SEM be a semantics satisfying GPPE *and* Elimination of Tautologies.

a) *Then:* $SEM(P) \subseteq \text{Min-MOD}_{\boxed{2\text{-val}}}(P)$ *for positive disj. programs P.*

I.e. any such semantics is already based on 2-valued minimal models.
In particular, *GCWA is the weakest semantics with these properties*.

b) *If SEM is non-trivial and satisfies in addition*[a] Isomorphy *and* Relevance, *then it* *coincides with GCWA on positive disjunctive programs*.

[a] See Chapter VII for the precise definitions of Relevance and Isomorphy.

Example 1.3 (Poole's Broken Arm)
Usually, a person's left arm is useable. But if the left arm is broken, it is an exception. The same statement holds for the right arm. Suppose that we saw Fred yesterday with a broken arm but we do not remember if it was the left or the right one. We also know that Fred can make out a cheque if he has at least one useable arm (he is ambidextrous) but that he is completely disabled if both arms are broken.

Here is the natural formalization:

$$
\begin{array}{lcl}
left\_use(x) & \leftarrow & \boxed{not}\ ab(left,x) \\
ab(left,x) & \leftarrow & left\_brok(x) \\
right\_use(x) & \leftarrow & \boxed{not}\ ab(right,x) \\
ab(right,x) & \leftarrow & right\_brok(x) \\
left\_brok(Fred) \boxed{\vee} right\_brok(Fred) & \leftarrow & \\
make\_cheque(x) & \leftarrow & left\_use(x) \\
make\_cheque(x) & \leftarrow & right\_use(x) \\
disabled(x) & \leftarrow & left\_brok(x), right\_brok(x)
\end{array}
$$

We expect that Fred is able to make out a cheque and that he is not (completely) disabled.

The problem is to rule out the unintended model where both arms are broken and Fred is disabled.

2 D-WFS

Before we can state the definition of D-WFS we have to extend our principles to disjunctive programs with default-negation. We abbreviate general rules

$$A_1 \vee \ldots \vee A_k \leftarrow B_1, \ldots, B_m, not\, C_1, \ldots, not\, C_n,$$

by

$$\boxed{\mathcal{A} \leftarrow \mathcal{B}^+, not\, \mathcal{B}^-}$$

where $\mathcal{A} := \{A_1, \ldots, A_k\}$, $\mathcal{B}^+ := \{B_1, \ldots, B_m\}$, $\mathcal{B}^- := \{C_1, \ldots, C_n\}$.

All our transformations generalize easily. For example:

Definition 2.1 (Positive and Negative Reduction)
Semantics $\mathcal{S}_{\mid\sim}$ *satisfies* **a)** Positive*, resp.* **b)** Negative *Reduction iff* $\mid\sim$ *is invariant under the following transformations:*

$$+\ \begin{array}{lcl} A \vee B & \leftarrow & \mathit{not}\ F \\ E & \leftarrow & A, \mathit{not}\ F \end{array} \quad \mapsto \quad \begin{array}{lcl} A \vee B & \leftarrow & \\ E & \leftarrow & A \end{array}$$

$$-\ \begin{array}{lcl} A \vee B & \leftarrow & \\ F \vee E & \leftarrow & C, \mathit{not}\ A, \mathit{not}\ B, \mathit{not}\ F \end{array} \quad \mapsto \quad \begin{array}{lcl} A \vee B & \leftarrow & \end{array}$$

Definition 2.2 (GPPE)

A semantics SEM satisfies GPPE if the following transformation does not change SEM:

Replace a rule $\mathcal{A} \leftarrow \mathcal{B}^+ \wedge not\ \mathcal{B}^-$ where $\mathcal{B}^+$ contains an atom B by the rules

$$\mathcal{A} \cup \left(\mathcal{A}_i \setminus \{B\}\right) \leftarrow \left(\mathcal{B}^+ \setminus \{B\}\right) \cup \mathcal{B}_i^+ \wedge not \left(\mathcal{B}^- \cup \mathcal{B}_i^-\right) (i = 1, \ldots, n)$$

where $\mathcal{A}_i \leftarrow \mathcal{B}_i^+ \wedge not\ \mathcal{B}_i^-$ $(i = 1, \ldots, n)$ *are all rules with* $B \in \mathcal{A}_i$.

$$\begin{array}{l} B \leftarrow not\ E \\ A' \vee B \leftarrow D, not\ C' \\ A \vee D \leftarrow B, not\ C \end{array} \quad \mapsto \quad \begin{array}{lcl} A \vee D \vee A' & \leftarrow & D, not\ C', not\ C \\ A \vee D & \leftarrow & not\ E, not\ C \end{array}$$

Defining a disjunctive version of WFS is easy:

Definition 2.3 (D-WFS)
There exists the weakest semantics satisfying positive and negative Reduction, GPPE, Elimination of Tautologies and non-minimal Rules. We call this semantics D-WFS.

As it was the case for WFS, our calculus of transformations is also confluent ((Brass and Dix 1995; Brass and Dix 1996)).

Theorem 2.4 (Confluent Calculus for D-WFS)
The calculus consisting of our four transformations is confluent and strongly terminating for propositional programs and fair sequences of transformations. I.e. we always arrive at an irreducible program, which is uniquely determined. The order of the transformations does not matter.

Therefore any program P is associated a unique normalform $\boxed{res(P)}$. The disjunctive wellfounded semantics of P can be read off from $\boxed{res(P)}$ as follows

$$\begin{aligned}\psi \in \text{D-WFS}(P) \quad \Longleftrightarrow \quad & \exists\, \mathcal{A} \subseteq \psi \text{ with } \mathcal{A} \leftarrow \mathbf{t} \in \boxed{res(P)} \quad \text{or} \\ & \exists\, not\, A \in \psi \text{ and } A \notin Head\_atoms(\boxed{res(P)}).\end{aligned}$$

Let us see how Example 1.3 is handled by D-WFS. Applying GPPE and Reduction gives us the following residual program (we consider just the *Fred*-instantiations):

$$
\begin{array}{lll}
left\_use(F) & \leftarrow & \boxed{not}\ ab(left,F) \\
ab(left,F) \boxed{\vee} right\_brok(F) & \leftarrow & \\
right\_use(F) & \leftarrow & \boxed{not}\ ab(right,F) \\
ab(right,F) \boxed{\vee} left\_brok(F) & \leftarrow & \\
left\_brok(F) \boxed{\vee} right\_brok(F) & \leftarrow & \\
make\_cheque(F) & \leftarrow & \boxed{not}\ ab(left,F) \\
make\_cheque(F) & \leftarrow & \boxed{not}\ ab(right,F)
\end{array}
$$

Therefore we derive $not\ disabled(F)$, because it does not appear in any head of the residual program. All the remaining atoms are undefined.

Two properties of D-WFS are worth noticing

- For positive disjunctive programs, D-WFS coincides with GCWA.
- For non-disjunctive programs with negation, D-WFS coincides with WFS.

3 DSTABLE

Unlike the wellfounded semantics, the original definition of stable models carries over to disjunctive programs quite easily:

Definition 3.1 (DSTABLE)
N *is called a* stable *model*[a] *of* P if and only if $N \in \mathit{Min\text{-}Mod}(P^N)$.

In the last definition P^N is the positive disjunctive program obtained from P by applying the Gelfond/Lifschitz transformation (as introduced before Definition 4.3 — its generalization to disjunctive programs is obvious).

[a] Note that we only consider Herbrand models.

Analogously to D-WFS the following two properties of DSTABLE hold:

- For positive disjunctive programs, DSTABLE coincides with GCWA.
- For non-disjunctive programs with negation, DSTABLE coincides with STABLE.

Of course, DSTABLE inherits the shortcomings of STABLE such as inconsistency and no goal-orientedness.

What about our transformations introduced to define D-WFS? Do they hold for DSTABLE? Yes, they are indeed true. The most difficult proof is the one for GPPE.

Theorem 3.2 (Characterization of DSTABLE)
Let SEM be a semantics satisfying GPPE, Elimination of Tautologies, *and* Elimination of Contradictions.
Then: $SEM(P) \subseteq STABLE(P)$.

Moreover, DSTABLE is the weakest semantics satisfying these properties.

DSTABLE is stronger than D-WFS as can be seen from Example 1.3. There we have exactly two stable models

1. $left\_use(F)$, $not\ ab(left, F)$, $ab(right, F)$, $not\ right\_use(F)$, $left\_brok(F)$, $not\ right\_brok(F)$, $make\_cheque(F)$, and $not\ disabled(F)$,

2. $right\_use(F)$, $not\ ab(right, F)$, $ab(left, F)$, $not\ left\_use(F)$, $right\_brok(F)$, $not\ left\_brok(F)$, $make\_cheque(F)$, and $not\ disabled(F)$.

In all of them, Fred is not disabled and (in contrast to D-WFS) can make out a cheque.

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.