

Loose notes and conclusions on a fixed point operator for rSM and rWFS

– DRAFT VERSION –

Luís Rodrigues Soares
March 27th, 2006

Introductory notes

This document includes a set of conclusions and steps taken in the direction of defining a fixed point operator for rSMs and rWFS. It is an account of the past 4 months of work and includes all ideas and intuitions tested so far. Its tone is rather informal and I'll be introducing all necessary concepts as they are required.

Part I

Four Months of Counter-Factual Programs

I start by describing the last four months of definitions based on generating counter-factual programs and calculating the stable models of the CFPs along with the original program.

1 The first definition¹

We were somewhere in December when the first stable idea for a fixed point operator came. Professor Moniz Pereira talked to me and Alexandre of the idea of Counter-Factual Programs (CFPs) and how they were used to help prove a certain literal in a RAA chain. This version of the definition of CFPs was unindexed in the default literals and throughout the rest of the document I'll refer to it as the unindexed version:

Definition 1 (Unindexed Counter-Factual Program P_{cf}) *Let P be a normal logic program with l rules in the form $r_k = H \leftarrow A_1, \dots, A_m, \sim B_1, \dots, \sim B_n$ ($m, n \geq 0$ and $1 \leq k \leq l$) and X a literal such that*

$$\exists_{r_k \in P} : X \in \{B_1, \dots, B_n\}$$

The counter-factual program $P_{cf}(X)$ regarding X and rule r_k in program P is the following transformation of P :

1. *Replace all non-negated literals A in all rules with the auxiliary literal A_ι ;*
2. *Remove from r_k all default literals $\sim X$;*
3. *Add to $P_{cf}(X_\iota)$ the rule $X \leftarrow X_\iota$*

¹Cova do Vapor version

The index used (ι , *iota*) is unique for this counter-factual program and literal X and is called the counter-factual index of $P_{cf}(X)$ and it identifies the literal we are trying to conclude.

The idea was that in order to prove a certain literal X by *reductio ad absurdum* (RAA) one would have to use a counterfactual program where we would assume $\sim X$. For example, consider a direct odd loop:

$$P = \{a \leftarrow x \\ x \leftarrow y \\ y \leftarrow \sim a\}$$

The CFP for a would be:

$$P_{cf}(a) = \{a \leftarrow a_1 \\ a_1 \leftarrow x_1 \\ x_1 \leftarrow y_1 \\ y_1 \leftarrow \}$$

In this program, we assume $\sim a$ as being true and try to see if a is a model of the program. We changed all literals to their indexed versions so we can later remove them from the program, i.e., the indexed literals are only necessary to create the RAA chain between $\sim a$ and a . The only thing provable from the CFP is actually a , because all indexed literals are not to be considered.

So, now we have:

$$\begin{aligned} \Gamma_P^r(I) &= \\ &= \Gamma_{P_{cf}(a) \cup P}(I) = \\ &= \{a_1, x_1, y_1, a\} = \\ &= \{a\} \end{aligned}$$

So the idea is that we should be able to use the CFPs of the atoms in an interpretation I , along with the original one, and if I was a stable model

(SM) of $P \cup P_{cfs}$, then I would be a rSM. We could then remove the indexed literals and only the unindexed ones were used in the result. Problems started precisely because when we are considering several literals in an interpretation they may interfere with each other's CFPs. For the odd loop of 3, $I = \{a, b\}$ is not a SM of $P \cup P_{cf}(a) \cup P_{cf}(b)$:

$$P = \{a \leftarrow \sim b \\ b \leftarrow \sim c \\ c \leftarrow \sim a\}$$

the CFPs for a and b would be:

$$\begin{array}{ll} P_{cf}(a) = \{a \leftarrow a_1 & P_{cf}(b) = \{b \leftarrow b_2 \\ a_1 \leftarrow \sim b & a_2 \leftarrow \\ b_1 \leftarrow \sim c & b_2 \leftarrow \sim c \\ c_1 \leftarrow \} & c_2 \leftarrow \sim a\} \end{array}$$

However,

$$\begin{aligned} \Gamma_{P_{cf}(a) \cup P_{cf}(b) \cup P}(I) &= \\ = \{b, b_1, c_1, a_2, b_2\} &= \\ = \{b\} \end{aligned}$$

From this observation, we came up with the idea that a_1 should be able to derive its truth value from a_2 , sort of like an auxiliary assumption, in order to give context to the fact that we were using several CFPs together. Symmetrically, b_2 could also depend on b_1 . Auxiliary assumptions undergone several versions, as they were more or less permissive and more or less *hacked*.

Definition 2 (Permissive Counter-Factual Auxiliary Assumptions operator α_{cf})

Let P be a normal logic program, $I = \{I_1, \dots, I_j\}$ a set of literals such that $1 \leq i \leq j$ and B a set of counter-factual programs obtained from the generation of a counter-factual program for each literal in I .

For each counter-factual program $P_{cf}(X)$ regarding a literal X that contains a rule in the form:

$$H_\iota \leftarrow A_1, \dots, A_m, \sim B_1, \dots, \sim B_n$$

such that

$$\exists_{P_{cf}(H)} : \text{the counter factual index for } P_{cf}(H) \text{ is } \kappa$$

introduce the rule $H_\iota \leftarrow H_\kappa$

Roughly speaking, this would give us the new result for $P_{cf}(a)$ and $P_{cf}(b)$:

$$\begin{array}{ll} P_{cf}(a) = \{a \leftarrow a_1 & P_{cf}(b) = \{b \leftarrow b_2 \\ a_1 \leftarrow \sim b & a_2 \leftarrow \\ b_1 \leftarrow \sim c & b_2 \leftarrow \sim c \\ c_1 \leftarrow & c_2 \leftarrow \sim a \\ a_1 \leftarrow a_2\} & b_2 \leftarrow b_1\} \end{array}$$

now giving us the intended result

$$\begin{aligned} \Gamma_{P_{cf}(a) \cup P_{cf}(b) \cup P}(I) &= \\ &= \{b, b_1, c_1, a_1, a_2, b_2\} = \\ &= \{a, b\} \end{aligned}$$

However, if $I = \{a, b, c\}$, which is not a revised stable model, our CFPs would be

$$\begin{array}{lll}
P_{cf}(a) = \{a \leftarrow a_1 & P_{cf}(b) = \{b \leftarrow b_2 & P_{cf}(c) = \{c \leftarrow c_3 \\
a_1 \leftarrow \sim b & a_2 \leftarrow & a_3 \leftarrow \sim b \\
b_1 \leftarrow \sim c & b_2 \leftarrow \sim c & b_3 \leftarrow \\
c_1 \leftarrow & c_2 \leftarrow \sim a & c_3 \leftarrow \sim a \\
a_1 \leftarrow a_3 & b_2 \leftarrow b_1 & c_3 \leftarrow c_1 \\
a_1 \leftarrow a_2\} & b_2 \leftarrow b_3\} & c_3 \leftarrow c_2\}
\end{array}$$

thus allowing $\Gamma_{P_{cf}(a) \cup P_{cf}(b) \cup P_{cf}(c) \cup P}(I) = \{a, b, c\}$.

So apparently, our auxiliary assumptions have to be more restrictive. Our next definition of the auxiliary assumptions added some defaults to each added rule; more specifically we would add to each auxiliary rule the defaults present in the original rule in program P , regarding the literal we were considering. For $I = \{a, b\}$, $P_{cf}(a)$ would become:

$$\begin{array}{l}
P_{cf}(a) = \{a \leftarrow a_1 \\
a_1 \leftarrow \sim b \\
b_1 \leftarrow \sim c \\
c_1 \leftarrow \\
a_1 \leftarrow a_2, \sim c\}
\end{array}$$

because as a_2 was present in $P_{cf}(b)$, it would have to respect the original defaults of the rule for head b in the original program².

This is the definition I brought with me to the first exam of KRR, before professor Moniz Pereira went to Indonesia. It became clear, during the exam, that the definition was very difficult to explain, and the intuitions were not very clear. Afterwards, I noticed that all the ideas I had presented still didn't work in some programs, which eventually took me to the next definition.

²The definition is very unintuitive and there is no point in presenting it here because it added no interesting notion to our work.

2 Afternoons at Praia da Torre

It was with the following program that I eventually came up with the most promising definition so far. The program that originated this definition was:

$$P = \{a \leftarrow \sim b \\ b \leftarrow \sim c, x \\ c \leftarrow \sim a \\ x \leftarrow \sim x, a\}$$

I'll skip all the steps I eventually went through, just to enumerate all the tricks I added in this definition:

Direct OLONs All direct single OLONs ($a \leftarrow \sim a$) were solved before anything else was done. This was because we already knew how these loops were handled, what their result was. Basically, if all our problems were like this, we would simply add a rule to the transformation which stated something like “remove $\sim X$ from the body of all rules with head X ”

Several CF programs for the same CF assumption If several $\sim X$ were present in the program, we would be creating a CFP for each one of them. This was because we viewed each $\sim X$ as one possible counterfactual path to X , so we would treat each path differently (we'll see later that this idea was indeed wrong).

Facts Additionally, we would not create CFPs for literal which were facts in the program. Although it was a trick, which if it wasn't present some programs would be giving more results than they should give, I still maintain that CFPs should not be created in this condition.

Auxiliary assumptions Finally, and probably most importantly, the auxiliary assumptions were now a lot simpler: a_1 could simply depend on b_2 if the rule for a_1 originally depended on $\sim b$, an idea of professor Moniz Pereira.

This is where the most promising definition of Γ^r was born. After some comments on the second exam of KRR I divided it in the following definitions :

Definition 3 (Counter-Factual Program P_{cf}) Let P be a normal logic program with l rules in the form $r_k = H \leftarrow A_1, \dots, A_m, \sim B_1, \dots, \sim B_n$ ($m, n \geq 0$ and $1 \leq k \leq l$) and X a literal such that

$$\begin{aligned} & \nexists_{r_k \in P} : r_k = X \leftarrow \\ & \quad \wedge \\ & \exists_{r_k \in P} : X \in \{B_1, \dots, B_n\} \end{aligned}$$

The counter-factual program $P_{cf}(X_\iota)$ regarding X and rule r_k in program P is the following transformation of P :

1. Replace all non-negated literals A in all rules with the auxiliary literal A_ι ;
2. Remove from r_k the default literal $\sim X$;
3. Add to $P_{cf}(X_\iota)$ the rule $X \leftarrow X_\iota$

The index used (ι , *iota*) is unique for this counter-factual program and this particular rule and is called the counter-factual index of $P_{cf}(X_\iota)$. It identifies both the rule used to try to conclude X and the literal we are trying to conclude. Hence the notation $P_{cf}(X_\iota)$.

Definition 4 (Counter-Factual Program Set β_{cf}) Let P be a normal logic program with rules in the form $r_k = H \leftarrow A_1, \dots, A_m, \sim B_1, \dots, \sim B_n$ ($1 \leq k \leq l$) and $I = \{I_1, \dots, I_j\}$ a set of literals such that $1 \leq i \leq j$. The set of all counter-factual programs of P regarding all literals in I and all rules in P is the set

$$\beta_{cf}^I(P) = \bigcup_{\iota} P_{cf}(X_\iota)$$

with ι being the program's counter-factual index, incremented as a new rule or literal is considered.

Definition 5 (Elegant Counter-Factual Auxiliary Assumptions operator α_{cf}) Let P be a normal logic program, $I = \{I_1, \dots, I_j\}$ a set of literals such that

$1 \leq i \leq j$ and B the counter-factual program set obtained from the application of $\beta_{cf}^I(P)$.

For each counter-factual program $P_{cf}(X_i)$ regarding a literal X_i that contains a rule in the form:

$$H_i \leftarrow A_1, \dots, A_m, \sim B_1, \dots, \sim B_n$$

such that

$$\begin{aligned} \exists_{Y \in \{B_1, \dots, B_n\}} : P_{cf}(Y) \in B \\ \wedge \\ Y \neq X \end{aligned}$$

introduce a copy of the rule regarding H_i such that $\sim Y$ is replaced with H_κ , κ being the counter-factual index identifying $P_{cf}(Y)$. Add this rule to $P_{cf}(X_i)$.

The application of $\alpha_{cf}^I(B)$ results in a program set with auxiliary assumptions added to each rule in B .

Definition 6 (Γ^r operator) Let P be a normal logic program and I an interpretation. The revised program P^r is the following sequence of operations:

1. Remove from all rules in P the default literal $\sim X$ such that X belongs to the head of the rule;
2. Create the counter-factual program set $B^r = \beta_{cf}^I(P_r)$ from interpretation I and the revised program P^r ;
3. Apply the counter-factual auxiliary assumptions operator $\alpha_{cf}^I(B^r)$ on B^r ;
4. Let $P^r = P^r \cup \alpha_{cf}^{B^r}(I)$.

On P^r apply the Γ operator and remove all counter-factual auxiliary literals X_i from the resulting set.

I eventually wrote several pages based on this definition and tried all the examples we had imagined so far and all was working well. Even though there were some tricks in between the results were promissing. I hand a hunch, however, that our auxiliary assumptions could be too permissive, and I was right. Consider program

$$P = \{y \leftarrow \sim b \\ b \leftarrow \sim c \\ c \leftarrow \sim a\}$$

whose only $rSM = \{y, c\}$. Let's create the CFPs for b and c , the only literals which appear in the head of a rule and negated in the body of other rules:

$$\begin{array}{ll} P_{cf}(b) = \{b \leftarrow b_1 & P_{cf}(c) = \{c \leftarrow c_2 \\ y_1 \leftarrow & y_2 \leftarrow b_2 \\ b_1 \leftarrow \sim c & b_2 \leftarrow \\ c_1 \leftarrow \sim a & c_2 \leftarrow \sim a \\ b_1 \leftarrow c_2\} & y_2 \leftarrow b_1\} \end{array}$$

The problem is that our CFPs allow us to prove that $\Gamma_P^r(\{b, c\}) = \{b, c\}$, even though not only c wasn't in an OLON, as b wasn't part of any kind of model (SM or rSM). c is provable from he main program, as a is not part of the model. And b comes out from the auxiliary assumption $b_1 \leftarrow c_2$. So the problem was, again, with the auxiliary assumptions...

3 Houston, we have a problem...

...was the subject of an email I sent to professor Moniz Pereira regarding this subject. The result, along with some discussion with Alexandre Pinto was a somewhat different approach to the definition which had the following main differences:

- Instead of generating only some CFPs, we generated all possible CFPs;
- All literals in our CFPs' rules, including the default ones, were indexed;

- The stable models of $P \cup P_{cfs}$ were the rSM of P . This involved trying to find out the right auxiliary literals of our transformation, in order to give the intended SM;
- The auxiliary assumptions were now defined differently: if a_1 depended on $\sim b_1$, we would add the dependency for b 's counterfactual program, $a_1 \leftarrow \sim b_1, \sim b_2$;
- Additionally, the counterfactual assumption of the CFP would be replaced by an odd loop, in order to switch on or off the odd loop.

Only to illustrate the definition, in the odd loop of 3 the CFPs would become:

$$\begin{array}{lll}
P_{cf}(a) = \{a \leftarrow a_1 & P_{cf}(b) = \{b \leftarrow b_2 & P_{cf}(c) = \{c \leftarrow c_3 \\
a_1 \leftarrow \sim b_1, \sim b_2 & a_2 \leftarrow b^- & a_3 \leftarrow \sim b_3 \\
b_1 \leftarrow \sim c_1 & b_2 \leftarrow \sim c_2, \sim c_3 & b_3 \leftarrow c^- \\
c_1 \leftarrow a^- & c_2 \leftarrow \sim a_2 & c_3 \leftarrow \sim a_3, \sim a_1 \\
a^- \leftarrow \sim a^+ & b^- \leftarrow \sim b^+ & c^- \leftarrow \sim c^+ \\
a^+ \leftarrow \sim a^- \} & b^+ \leftarrow \sim b^- \} & c^+ \leftarrow \sim c^- \}
\end{array}$$

We can see the main differences: the auxiliary assumptions are now embedded in the main rules of the program, all literals have indexes and the odd loop can be switched on or off by considering wither b^- or b^+ as being part of the model. All examples appeared to be working well until the following program:

$$\begin{array}{l}
P = \{b \leftarrow \sim a, c \\
c \leftarrow a \\
a \leftarrow \sim b\}
\end{array}$$

which has $rSM_1 = \{a, c\}$ and $rSM_2 = \{b\}$.

$$\begin{aligned}
P_{cf}(a) = \{ & a \leftarrow a_1 \\
& b_1 \leftarrow a^-, c_1 \\
& c_1 \leftarrow a_1 \\
& a_1 \leftarrow \sim b_1 \\
& a^- \leftarrow a^+ \\
& a^+ \leftarrow a^- \}
\end{aligned}$$

$$\begin{aligned}
P_{cf}(b) = \{ & b \leftarrow b_2 \\
& b_2 \leftarrow \sim a_2, c_2, \sim a_1 \\
& c_2 \leftarrow a_2 \\
& a_2 \leftarrow b^- \\
& b^- \leftarrow b^+ \\
& b^+ \leftarrow b^- \}
\end{aligned}$$

$$\begin{aligned}
P_{cf}(c) = \{ & c \leftarrow c_3 \\
& b_3 \leftarrow \sim a_3, c_3 \\
& c_3 \leftarrow a_3 \\
& a_3 \leftarrow \sim b_3 \\
& c^- \leftarrow c^+ \\
& c^+ \leftarrow c^- \}
\end{aligned}$$

With our transformation, b would never make it to the model because it would end up depending on both a_2 and $\sim a_2$. The problem was that, even though the chain of support for b demands a to be true, in order to get to $\sim b$, the chain also demanded $\sim a$ to be true. However, $\sim a$ would eventually become true as soon as we proved that there was in fact a chain of support between b and $\sim b$ and hence all the literals in the chain would have to be false.

The solution we came up with was to replace all default literals $\sim X_1$ in the body of all rules by an extended literal X_1^* , being that $X_1^* \leftarrow \sim X_1$ and $X_1^* \leftarrow \sim X$. This allowed us to, while having a_2 in the stable model,

therefore cutting one of the rules for a_2^* , keep the other one, because a was not part of the model.

It was an interesting idea, and a necessary one, but it was incomplete. With our well known odd loop of 3, we couldn't get the models we wanted. A solution for this was discussed, which involved counting the number of negations we went through as we were descending through the dependency graph for each literal (the problem was here...), but these ideas still weren't enough³.

Here's why: not only did the auxiliary assumptions kept some odd loops active in the transformed program, as they prevented some things from being proved. Let's look at the odd loop of 3. Our CFPs are⁴:

$$\begin{array}{lll}
 Pcf(a) = \{a \leftarrow a_1 & Pcf(b) = \{b \leftarrow b_2 & Pcf(c) = \{c \leftarrow c_3 \\
 a_1 \leftarrow b_1^*, \sim b_2 & a_2 \leftarrow b^- & a_3 \leftarrow \sim b_3 \\
 b_1 \leftarrow \sim c_1 & b_2 \leftarrow \sim c_2^*, \sim c_3 & b_3 \leftarrow c^- \\
 c_1 \leftarrow a^- & c_2 \leftarrow \sim a_2 & c_3 \leftarrow \sim a_3^*, \sim a_1 \\
 b_1^* \leftarrow \sim b_1 & c_2^* \leftarrow \sim c_2 & a_3^* \leftarrow \sim a_3 \\
 b_1^* \leftarrow \sim b & c_2^* \leftarrow \sim c & a_3^* \leftarrow \sim a \\
 a^- \leftarrow \sim a^+ & b^- \leftarrow \sim b^+ & c^- \leftarrow \sim c^+ \\
 a^+ \leftarrow \sim a^- \} & b^+ \leftarrow \sim b^- \} & c^+ \leftarrow \sim c^- \}
 \end{array}$$

Now, if $I = \{a, b\}$, a_1 must be part of the interpretation, because the only way to prove a is through its CFP. However, if a_1 is part of the interpretation, c_3 will be false, thus allowing b_2 to become true. But b_2 can't be, or else the rule for a_1 will be cut. Even if we only generated the CFPs for the literals present in the model, the problem still maintained: b_2 would depend solely on c_2^* which, in order to become false, would demand both c and c_2 to be true. But c can't be true, particularly if we are testing $I = \{a, b\}$...

The Houston definition was not ready yet...

³I never wrote any formal definition of these ideas, since the Houston problem, because all the problems appeared during testing of programs, no definition ever succeeded in all programs

⁴We are now only generating the star rules for the even transitions over negation, starting from the literal we are trying to prove. this was the very last changing I discussed with Professor Moniz Pereira.

4 Some initial conclusions

More or less at the same time I was realizing all the previous problems, I started writing the several things I thought that contributed to the failure of all the previous definitions, as well as all the things I liked/disliked. Here's a brief account on them:

How to generate CFPs Two methods were tested to generate CFPs. The first one, used until the Houston problems, used no indexes on the default literals. From the Houston definition onward the definition started using indexes on the default negated literals. From these two methods, I prefer the approach of the Houston definition, i.e., generate indexes on all literals. This is because it gives a better image of the reasoning we are trying to apply: our CFP is a test program, to see if, by assuming a certain $\sim X$ we eventually get to X . “get to” is a way of saying that there has to be a chain of support from the CF assumption to the literal we are trying to prove. It's easier to see this chain if all literals are numbered. The rest of the method to generate CFPs is equal on all approaches. The CF assumption is removed from the CFP and a rule is added to the CFP which connects it to the original program. This is a guarantee that this CFP only allows its CF assumption to be concluded.

When to generate CFPs The idea of generating CFPs is uniquely to allow some literal to be proven by RAA reasoning. On some instances of the definition we generated all possible CFPs possible, i.e., for all literals present in the program. On other instances, we only generated CFPs for the literals present in the interpretation. This could be constrained even further by demanding the literals for which we generated CFPs had to appear at least once in the head of some rule, and at least once negated in the body of some rule. These two conditions are essential conditions for any literal to be provable by RAA. At least these two conditions have to be satisfied.

Auxiliary assumptions How should we connect the several CFPs? Well, I really have no conclusion here. I can say that the last approach was the one more intuitive for me, and the one that appeared to work the best. I believe that, ideally, the connection between the CFPs should be implied from the original program, i.e., we shouldn't have to add any rules to the CFPs. However, so far, no perfect way for doing this was defined...

Tricks On the several tricks we've tested, I believe on the idea of not generating CFPs for facts. The reason is simple: we want to generate as

few CFPs as possible. So if something is a fact it is already proved. Additionally, I also believe that all programs can be simplified by solving all the direct OLONs, although, as professor Moniz Pereira said, this is only an optimization, it should not be mandatory in the definition. The trick of generating several CFPs, for each CF assumption was obviously wrong. When we assume some $\sim A$, all $\sim A$ s of the program are assumed, we can't assume some and don't assume others.

Using Stable Models on the transformed program The use of stable models to see if a certain literal is provable by RAA is an idea I don't find very intuitive, precisely because it doesn't reflect the original idea that an atom is provable by RAA if there is a chain of support between the atom and its default negation. And this was probably the thing that was annoying me the most in all our previous attempts.

This concludes the first part of this document. All the definitions we have tested so far, why they have failed and where they have failed.

Part II

An alternative definition

Here I describe the last idea I had for a fixed point operator for revised stable models, not based on CFPs.

5 Origin of the idea

About the same time I wrote the previous text Alexandre talked with me of an idea he once had for an operator, but which he never explored. It basically consisted in adding some rules to the program that reflected the conditions the program was stating. For example in our loop of 3, we can see that the main condition for having a and b is not to have c . So apparently, we could add $a \leftarrow \sim c$ to the program, and this would allow us to have $I = \{a, b\}$ as a stable model of P (symmetrically for the other interpretations). The idea was interesting but the reasons didn't look very intuitive so I didn't thought of it for much longer.

A couple of days later I was trying a different approach for the problem which consisted of the following intuition: if we could get rid of the OLONs and ICONs, the Γ operator would have no problem in calculating the models (revised or not). In other words, it would be handy if we could transform OLONs and ICONs into ELONs. How can we do that? For example, for the following program

$$P = \{a \leftarrow b \\ b \leftarrow c \\ c \leftarrow \sim a\}$$

it would be necessary to have a rule with a in the head. The only rule we could invert was the last one, becoming $a \leftarrow \sim c$. Curiously, this was very close to the idea Alexandre discussed with me, although it was seen from a different point of view. I showed him some preliminary results, looking promising and we went on trying the idea. Initially it was a rather naive approach, we would invert every rule in the program, which had no justifi-

cation possible. Obviously after a few programs the idea stopped working, and I put it to rest for a few days.

6 Häagen-Dazs at Belém

The Häagen-Dazs shop near Torre de Belém, in Lisbon, is a great place to have breakfast and ideas. It was there, on a Saturday morning, that I returned to my last suggestion of an operator for rSMs and rWFS, using the previous intuitions and some new ideas I explore next.

I eventually returned to approach I had discussed with Alexandre because I thought that there had to be some reason for it to work in some programs. I then did the oposite exercise (using the previous program): if I wanted to prove a , what should be the body? Well, if we look at the loop, we can see that the only safe condition to have a is not to have c . Why? Because, being the tail of the odd loop, as a is true, the rest of the chain will be false. In particular, b and c will also be false but it could be the case that we had some default literals along the chain (as in the odd loop of 3), so the safest condition for a being true is the head of the rule that depends on $\sim a$ being false, i.e., $a \leftarrow \sim c$. Now if c is false, so will b , and so will a .

There was, however, a catch: when are we allowed to add such rules? Well, if and only if there is a chain of support between $\sim a$ and a . This is precisely the principle behind all these semantics: if a literal can be proved by RAA it should be added to the program, along with the conditions it must respect. In the case of the previous simple example, the condition is simple: a will be true if c is false. What about other programs? For example:

$$P = \{b \leftarrow \sim a, c \\ c \leftarrow a \\ a \leftarrow \sim b\}$$

which has $rSM_1 = \{a, c\}$ and $rSM_2 = \{b\}$. To begin with, we can see that a is not in an OLON. It's easy to see this if we draw the chain of support, starting with $\sim a$.

DESENHO AQUI PARA A ARVORE DO A

I put c in brackets to say that c is a precondition to move on to the next

branch of the tree. Well, if a is not in an OLON, it must be proven by the classical notion of support, or it doesn't belong to any model (revised or otherwise). c , never appears negated in the body of any rule, so it can never be proven by RAA reasoning. Let's turn our attention now to b .

DESENHO AQUI PARA A ARVORE DO B

$\sim b$ will eventually lead to b , meaning it is in an odd loop. It has a precondition that is $\sim a$ must be true. So if we test any interpretation which includes b , we must add the rule $b \leftarrow \sim a$. Note that $\sim a$ is not only the precondition in brackets but also the tail of the chain of support. However, this idea of drawing the chain of support for each literal, starting with $\sim X$ to see if we reach X is not very beautiful so I put the intuition behind this in an operator. I called it Δ operator (*delta*, which starts with "d", as the word "derivation"). Intuitively it works somewhat like this:

Definition 7 (Δ operator⁵) *Let P be a NLP with rules r in the form $r = H \leftarrow A_1, \dots, A_m, \sim B_1, \dots, \sim B_n$. Δ operator has the following behaviour:*

1. $\Delta(\{X\}) = \{H\}, \exists_{r \in P} : r = H \leftarrow X$, being X a literal (default or not);
2. $\Delta(\{X\}) = \{H, [Y]\}, \exists_{r \in P} : r = H \leftarrow X, Y$, being X and Y literals (default or not);
3. $\Delta(\{\sim X\}) = \{\sim H\}, \exists_{r \in P} : r = H \leftarrow X$, being X a literal (default or not);
4. $\Delta(\{\sim X\}) = \{\sim H, [Y]\}, \exists_{r \in P} : r = H \leftarrow X, Y$, being X and Y literals (default or not);
5. $\Delta(\{X\}) = \{H_1, H_2\}, \exists_{r_1, r_2 \in P} : r_1 = H_1 \leftarrow X$ and $r_2 = H_2 \leftarrow X$, being X a literal (default or not);
6. $\Delta(\{X_1, \dots, X_n\}) = \text{create a copy of the set as it is so far, and continue with the two instances separated.}$

So, this operator generates ordered sets, which start with the default literal we are assuming to test our RAA chain. Everytime we reach a point where we have a conjunction, we simply put the head of that conjunction in brackets. We don't need to go down that conjunction because this operator's purpose is not to prove a certain literal – its objective is simply to show the literals we have to go through to reach a certain X starting with $\sim X$. If, at any point, we reach a disjunction, i.e., two rules, with the same literal in the

tail, we generate two derivation trees, which are equal until the disjunction point, and continue from that point onward.

I'll now show how this behaves in some programs, showing not only the derivation trees, but also the result of the Δ operator. These programs are generally enhanced versions of the programs available in the previous tests set Alexandre gave me.

$$\begin{aligned}
 P_1 = \{ & b \leftarrow y \\
 & y \leftarrow \sim a, c \\
 & c \leftarrow a \\
 & a \leftarrow x \\
 & x \leftarrow \sim b \}
 \end{aligned}$$

$$\begin{aligned}
 P_2 = \{ & a \leftarrow x \\
 & x \leftarrow \sim a, y \\
 & y \leftarrow \sim b \\
 & b \leftarrow z \\
 & z \leftarrow \sim b, w \\
 & w \leftarrow \sim a \}
 \end{aligned}$$

$$\begin{aligned}
 P_3 = \{ & a \leftarrow \sim b, \sim c \\
 & b \leftarrow \sim a, \sim c \\
 & c \leftarrow \sim b, \sim a \}
 \end{aligned}$$

$$\begin{aligned}
 P_4 = \{ & a \leftarrow x \\
 & x \leftarrow y \\
 & y \leftarrow \sim a \\
 & y \leftarrow \sim b \}
 \end{aligned}$$

$$P_5 = \{a \leftarrow x$$

$$x \leftarrow y, \sim a$$

$$y \leftarrow \sim a\}$$

$$P_6 = \{a \leftarrow x$$

$$x \leftarrow y, \sim x$$

$$y \leftarrow \sim a\}$$

$$P_7 = \{a \leftarrow \sim b$$

$$b \leftarrow \sim c, x$$

$$c \leftarrow \sim a$$

$$x \leftarrow a, z$$

$$z \leftarrow \sim x\}$$

$$P_8 = \{a \leftarrow x, y$$

$$x \leftarrow z$$

$$z \leftarrow \sim a$$

$$y \leftarrow w$$

$$w \leftarrow \sim a\}$$

$$P_9 = \{a \leftarrow \sim a, \sim b$$

$$b \leftarrow \sim b, d$$

$$d \leftarrow \sim a\}$$

$$P_{10} = \{a \leftarrow x$$

$$x \leftarrow \sim a$$

$$b \leftarrow \sim a$$

$$c \leftarrow \sim b$$

$$d \leftarrow \sim c\}$$

$$P_{11} = \{a \leftarrow \sim b \\ b \leftarrow a\}$$

$$P_{12} = \{y \leftarrow \sim a \\ a \leftarrow \sim b \\ b \leftarrow \sim c\}$$

$$P_{13} = \{a \leftarrow \sim x, \sim y \\ x \leftarrow \sim z \\ z \leftarrow \sim a \\ y \leftarrow \sim w \\ w \leftarrow \sim a\}$$