Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics
Department of Applied Informatics

Complexity of Revised Stable Models

Michal Malý

Advisors:
Prof. Luís Moniz Pereira,
Assoc. Prof. PhDr. Ján Šefránek, PhD.

Master's Thesis
Lisbon, Bratislava 2006, 2007

Declaration

Hereby I declare that this master's thesis is the result of my own work, except where otherwise indicated. I have only used the resources given in the list.

Copyright © 2006, 2007 Michal Malý

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Acknowledgments

I wish to thank Prof. Pereira and Assoc. Prof. Šefránek for their support, a careful supervision and bright ideas. I wish to thank Mr. Pinto for his support and useful explanations. I wish to thank Martin Baláž for his notes and questions which helped to improve this work.

Contents

Chapter 1

Introduction

A beginning is a delicate time, tells us Princess Irulan from Frank Herbert's Dune. When I began to learn about Stable Models at the Knowledge representation and inference course led by Assoc. Prof. Šefránek, I had a lot of (maybe stupid) questions and thoughts. This was a delicate time for me: For a while, I considered Stable models counterintuitive, because such a simple program like $a \leftarrow \sim a$ has no model.

Of course, at the beginning, one does not possess an adequate knowledge of the topic. This is usually a disadvantage, but brings a possibility to formulate novel ideas, uninfluenced by the common knowledge. When one learns what others already know, it is not so easy to invent something new. This is my case: I had my feeling, but I did not try to evolve this idea further.

This is the primary reason why I was excited when I heard about Revised Stable Models. They bring us the possibility to infer the way I missed in Stable Models, using so called *reductio ad absurdum*. Attractive is that Revised Stable Models semantics is an extension of Stable Models semantics, what means, that every Stable Model is a Revised Stable Model. Later we will mention a few other good properties of Revised Stable Models.

I was interested in computational complexity, too, so I considered a good idea to connect these two interests and investigate the complexity of Revised Stable Models. This issue was not covered by the authors of the idea, and was suitable for my Master's Thesis.

During the first 4 months of the work, I visited Centro de Inteligência Artificial – CENTRIA, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Portugal, thanks to the Erasmus education program. I could cooperate directly with the authors of the idea, Luís Moniz Pereira and Alexandre Miguel Pinto.

The flow of the work was not straightforward. Many ways were blind and not every result was correct. But this is not a bad thing, it is an essential part of every research. The possibility of a collaboration with experts was for me an additional benefit of the work.

Chapter 2

Reductio ad absurdum

2.1 Classical *reductio ad absurdum*

Wikipedia [5] says: *In formal logic,* reductio ad absurdum *is used when a formal contradiction can be derived from a premise, allowing one to conclude that the premise is false. If a contradiction is derived from a set of premises, this shows that at least one of the premises is false, but other means must be used to determine which one.*
Formally,

if $S \cup p \vdash \bot$ then $S \vdash \neg p$
or if $S \cup \neg p \vdash \bot$ then $S \vdash p$.

Encyclopædia Britannica [2] states about *reductio ad absurdum* the following: *in logic, a form of refutation showing contradictory or absurd consequences following upon premises as a matter of logical necessity.*

2.2 *Reductio ad absurdum* in the logic programming

We can discuss if the concept of *reductio ad absurdum* is even possible in logic programming. Here we have no procedural way to put premises, and then indicate their absurd consequences. We have only one way of indicating a contradiction in the program: to have no model at all (and this is exactly what Stable Models do when they encounter a program like $a \leftarrow \sim a$). But then we cannot continue with our derivation and declare the premise invalid, we are bound to the "no model" result.
The two-step procedure (first derive a contradiction, then declare a premise false) is apparently unsuitable for logic programming.
However, we feel somehow, that it is proper to call *reductio ad absurdum* an (from the programmer's point of view) one-step procedure of computing a model, if it brings us a similar result as the aforementioned formal procedure in classical logic.

2.3 Motivation for *reductio ad absurdum* in Stable Models

To excuse introducing a new semantics is a hard task. Generally, to excuse introducing a new thing is a hard task, because people will stick to their habits and traditions, and until we supply them a good reason to switch, they will not accept the new. I do not feel competent to judge if *reductio ad absurdum* is a good reason for logic programmers to switch. Maybe not. But maybe sometimes in the future, for some project, the *reductio ad absurdum* will be so important, that programmers will choose this way. Even if not, we can always excuse ourselves simply by saying "we are scientists, our research is not worth because its practical use, but for its beauty."

Reductio ad absurdum is used in math-, law- and even in common-life- argumentation. People (e.g. me:-) who saw Stable Models for the first time wonder why the simple program $a \leftarrow \sim a$ has no Stable Model. Of course, many people (especially advanced logic-programmers) consider this property as a basic tool for programming (e.g. introducing integrity constraints).

We can solve this problem by using an explicit construction in the language, e.g. *falsum* literal. In my opinion, *reductio ad absurdum* in the logic programming can be closer to human thinking and to the way people argue.

2.3.1 Motivational example

Three gay friends are deciding which night club they want to entertain this night in. Each has his favorite club, but is willing to make a compromise. They expressed their views in this way[1]:

The first one says: *If we are not going to dance in Apollon, let's have a coffee in Barbaros.*
The second one argues: *Well, if we are not going to Barbaros, let's see the boys in Crater.*
And the third one: *But, when we are not going to Crater, I want to go to Apollon.*

We are able to formally rewrite their opinions into this small program:

$b \leftarrow \sim a$
$c \leftarrow \sim b$
$a \leftarrow \sim c$

But what then? Using Stable Models will make no good. Use of *reductio ad absurdum* is necessary.

[1] The mentioned three Bratislava gay clubs are Barbaros at Vysoká 20, Kráter at Vysoká 14 and Apollon at Panenská 24.

2.4 *Reductio ad absurdum* in Minimal Model semantics

In the example above, we can realize that use of (Minimal) Models semantics has the expected result of using *reductio ad absurdum* like in classical logic. This is due to the definition of a "model": It satisfies all rules of the program, what makes *reductio ad absurdum* automatically valid. A model definition has no procedural part. For simple programs, we could be satisfied with this. If we expect larger programs, we might want to have a "stable" semantics. (In chapter 7, we will see how Revised Stable Models and Minimal Models are connected).

2.5 How to introduce *reductio ad absurdum* in SM

First we'll consider *reductio ad absurdum* for minimal models.
We can realize that

a) minimal models does satisfy the *reductio ad absurdum* condition but

b) minimal models do not share the "stability" of SM,

so we obviously need something between (in the set meaning) minimal models and SM that will ensure the stability.

What does the statement a) mean?

For each program P, a set S and an atom x holds:
if $P \cup S \cup x \vdash \perp$ then
no minimal model $M \supseteq S$ exists so that $x \in M$.

or similarly for $\neg x$
if $P \cup S \cup \neg x \vdash \perp$ then
for every minimal model $M \supseteq S$ holds $x \in M$.

The reader probably noticed we are using the symbol $\vdash$ in two slightly different meanings: In the former relation as "by using derivation rules" and in the latter in form $\vdash \perp$ as "is not a model". I think there is an analogy: In the logic we can think of $T \vdash \perp$ as "T is not consistent". Moreover we can think about $\vdash$ used in connection with (minimal) models of a program P as "by using rules of P". We can think of the rules as of implications. Due to the transitivity of implication, we could translate this as $\vdash \equiv \bigcup_{i>0} \rightarrow^i$.

And what does it mean "the stability" (in the b) statement)?

We will continue with the analogy. When in minimal models we are using $\rightarrow$ as a basic inference rule, in Stable models this is different. The basic operator is Γ [2]. We will see this operator is important in the definition of Revised Stable Models.

Finally, we need some condition to ensure that *reductio ad absurdum* does not play against us. This condition we can formulate in this informal way: For each consistent S, when we add a new atom, it will not broke the consistency. More formally it is formulated in the next chapter in the definition of RSM.

2.6 Examples

Taken over from [13].

2.6.1 Minimal Models which are not "stable"

$a \leftarrow \sim b$
$t \leftarrow a, b$
$k \leftarrow \sim t$
$b \leftarrow \sim a$
$i \leftarrow \sim k$

The minimal models are: $\{a, k\}, \{b, k\}, \{a, t, i\}, \{b, t, i\}$. The second two are suspect: how we can say t is true, when the rule for t is not applicable, because a, b are never true at the same time? Said in another way, we cannot get $\{a, t, i\}, \{b, t, i\}$ by iteration of Γ.

2.6.2 Another example

$a \leftarrow \sim a$
$b \leftarrow \sim a$
$c \leftarrow \sim b$
$d \leftarrow \sim c$
$e \leftarrow \sim e$

Here for the MM $\{a, b, d, e\}$ we cannot apply the second rule and get so the atom b. Intuitively we feel we have to take care about such situations. Formal definition we show in the next chapter.

[2] $\Gamma_P(M) = M(GL(P, M))$ – the least model of GL-transformation of P modulo M

Chapter 3

Revised Stable Models

Definition 1 (Gelfond-Lifschitz Γ_P operator [9])**.** Let P be a NLP and I a 2-valued interpretation. The GL-transformation of P modulo I is the program P/I, obtained from P by performing the following operations: - remove from P all rules which contain a default literal $notA$ such that $A \in I$
- remove from the remaining rules all default literals
Since P/I is a definite program, it has a unique least model J: Define $\Gamma_P(I) = J$.

Definition 2 (Stable Models)**.** Stable Models are the fixpoints of Γ_P.

As a shorthand notation, let $WFM(P)$ denote the positive atoms of the Well-Founded Model of P, that is $WFM(P)$ is the least fixpoint of operator Γ_P^2 [14], i.e. Γ_P applied twice.

We will now define Revised Stable Models. The definition comes from the paper [13]. First we introduce the notion of *sustainability*.

Definition 3 (*Sustainable* Set [13])**.** Intuitively, we say a set S is *sustainable* in a NLP P iff any atom a in S does not go against the well-founded consequences of the remaining atoms in S, whenever, $S \setminus \{a\}$ itself is a *sustainable* set. The empty set by definition is *sustainable*. Not going against means that atom a cannot be false in the Well-Founded Model of $P \cup S \setminus \{a\}$, i.e., a is either true or undefined. That is, it belongs to set $\Gamma_{P \cup S \setminus \{a\}}(WFM(P \cup S \setminus \{a\}))$. Formally, we say S is *sustainable* iff

$$\forall_{a \in S} S \setminus \{a\} \text{is sustainable} \Rightarrow a \in \Gamma_{P \cup S \setminus \{a\}}(WFM(P \cup S \setminus \{a\}))$$

If S is empty the condition is trivially true.

The definition of Revised Stable Model is the following

Definition 4. Revised Stable Model ([13])
Let $RAA_P(M) = M - \Gamma_P(M)$. M is a Revised Stable Model of a NLP P, iff:

1. M is a minimal classical model of P, with $\sim$ interpreted as classical negation
2. $\exists_{\alpha \geq 2} \Gamma_P^\alpha(M) \supseteq RAA_P(M)$
3. $RAA_P(M)$ is a *Sustainable* Set

3.0.3 Good properties of Revised Stable Models

The paper [13] describes other good properties of Revised Stable Models: relevance, cumulativity, and existence. Description of the first two is out of scope of this work. The existence property means that each program has a Revised Stable Model.

3.0.4 Sustainability notion

The notion of sustainability is at the core of the Revised Stable Models definition. We require that the $RAA_P(M)$ for a given interpretation M to be *Sustainable* in order for M to possibly be a Revised Stable Model of P (this corresponds to the third condition of the definition of Revised Stable Model).

When we thoroughly analyze the meaning of the $RAA_P(M)$ set we understand that it is the subset of atoms of M which are necessary by *reductio ad absurdum* reasoning in P, under the context of the remaining atoms of M, i.e., $M - RAA_P(M) = \Gamma_P(M)$.

As seen in [13], *reductio ad absurdum* reasoning is required in only two cases: when Odd Loops Over Negation and/or Infinite Chains Over Negation are present in the Normal Logic Program P. As explained in [13], in a normal logic program, we say we have a loop when there is a rule dependency call-graph path that has the same literal in two different positions along the path - meaning that the literal depends on itself. An Odd Loop Over Negation (OLON) is a loop such that the number of default negations in the rule dependency graph path connecting the same literal at both ends is odd.

Example 5. Example of an Odd Loop Over Negation
Let's have the following program:

$x \leftarrow \sim x$
$a \leftarrow \sim b$
$b \leftarrow \sim c$
$c \leftarrow \sim a$

In this program we have two Odd Loops Over Negation: the first one is an OLON over x (x directly depends on $\sim x$), and the other is an OLON over the three atoms a, b, and c (each one of a, b, and c depends on its own negation through the two other atoms — there is an Odd number of Default Negations in the dependency graph from a to a, from b to b and from c to c).

The reader can check that the computation of Revised Stable Model complies with the intuition we used in examples in the previous chapter.

3.1 More about sustainability[1]

The notion of sustainability is difficult to track, because its definition is recursive[2]. Now I will try to help reader to understand this concept in an imaginative way of lattice-like diagrams. I have used this visualization during my work and it helped me to understand the concept. I have been also able to find a counterexample to a conjecture/belief about sustainability held by original authors of RSM.

3.1.1 Visualization of sustainability

Figure 3.1.1 is an image of all subsets of the set of atoms of a program. The sets are arranged in a hierarchical way, forming so called "power lattice". The downwards direction means "subset of", the direction upwards is the direction of "superset of".
Edges connecting the circles are colored in two ways, depending on the condition $a \in \Gamma_{P \cup S \setminus \{a\}}(WFM(P \cup S \setminus \{a\}))$ from the definition of sustainability. The a in the condition is the atom which we have to add to the subset to gain the superset. The gray circle means that the set is sustainable.
Now we can formulate the definition sustainability in a graphical, more imaginable way: A circle is gray, if all its gray children are connected with him by a black line.

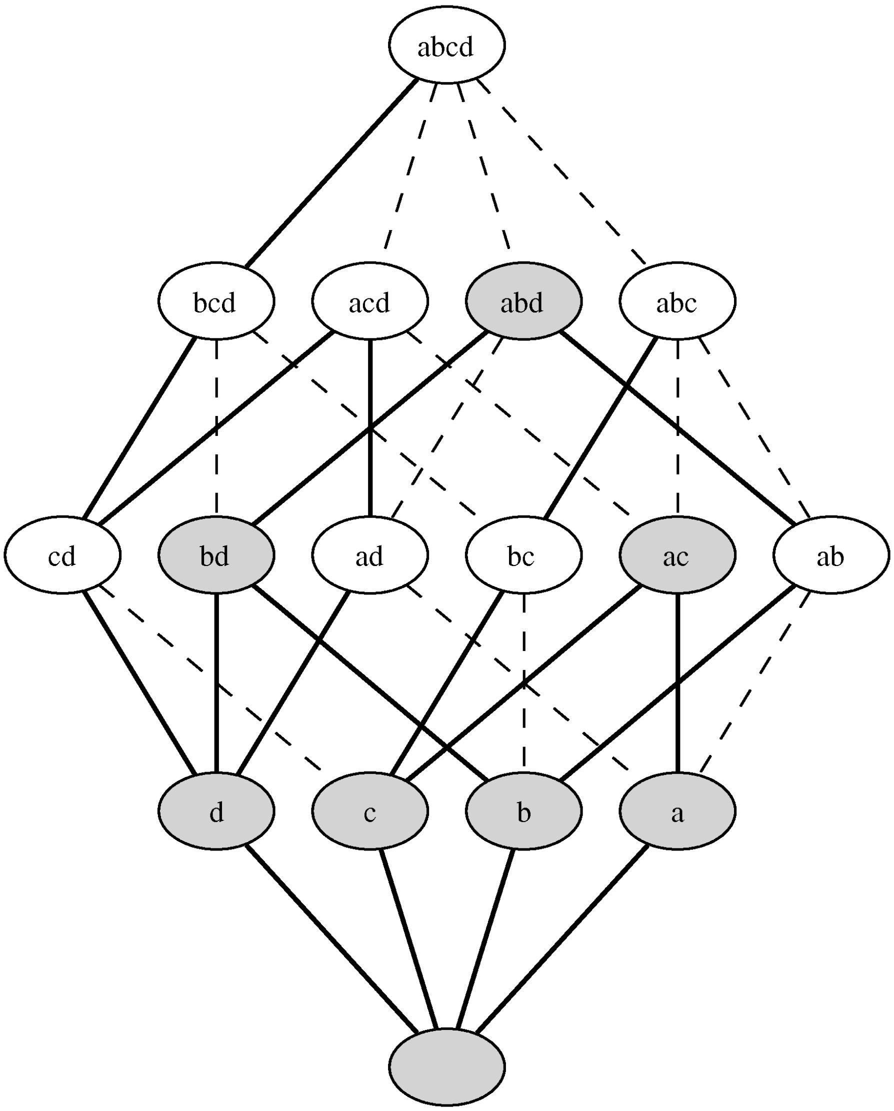
3.1.2 Special cases of sustainability

Let's look at the figure 3.1.2. Here is an example of sustainable set, $\{a, b, c\}$. This set is sustainable, because none of the sets $\{a, b\}, \{b, c\}, \{a, c\}$ is sustainable. We cannot find a path ("going through black lines and gray circles") from the empty set to $\{a, b, c\}$. It is an example of sustainable set, which can never be RAA(M) for some Revised Stable Model M.[3]

[1] All graphs in this section all plotted by graphviz package, see [8]

[2] I personally consider this complex notion the main barrier for conveying the idea of RSM among experts.

[3] This is intuitively clear. We tried to prove the conjecture, that for every rSM(M), the RAA(M) set has to have such a path but we have not finished the formal proof as of the time.

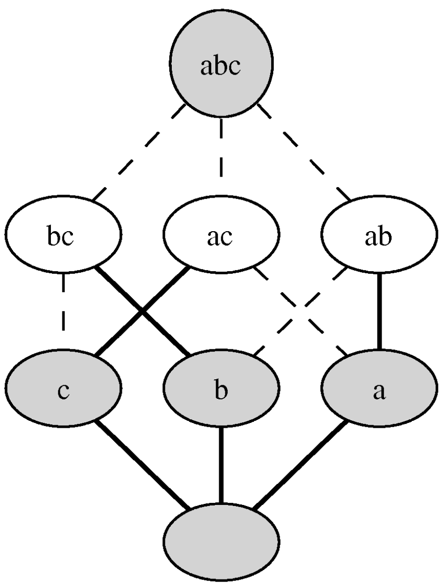


(a) The visualization

$$a \leftarrow \sim a$$
$$b \leftarrow \sim a$$
$$c \leftarrow \sim b$$
$$d \leftarrow \sim c$$

(b) The program

Figure 3.1: Lattice-like sustainability visualization for a program



$a \leftarrow \sim b$
$b \leftarrow \sim c$
$c \leftarrow \sim a$

(a) The visualization (b) The program

Figure 3.2: Special case of sustainability

Chapter 4

Complexity issues

4.1 Standard complexity questions

When we ask for complexity in logic programming, we usually think about fourt typical basic problems. Their formulation follows:

Complexity checking. Decide, whether the given set of atoms is a model of the program.

Problem Q1. Decide, whether the given program has a model.

It has been shown that every program has a rSM [13]. So this problem is trivial. (Compare with Stable models, where this question is NP-complete [7]).

Problem Q2 (Brave reasoning). Decide, whether the given program has a model in which a given literal is true.

Problem Q3 (Cautious reasoning). Decide, whether in all models of a given program is the given literal true.

4.1.1 Brave reasoning

It has been shown that for SM, brave reasoning problem is NP-complete [10].

We cannot use the proof of NP-hardness of Q2 for SM to prove NP-hardness of Q2 for rSM, because the proof (reduction from kernel of graph) is based on non-existence of SM of program with OLON. But we can construct another proof – reduction from 3SAT.

Lemma 6. *Problem Q2 for revised stable models is NP-hard.*

Proof. Let's have instance of 3SAT, a formula E, and suppose we have an oracle which solves $Q2$ in polynomial time. From formula E we construct a program P, so that P will have a rSM where the atom $satisfiable$ will be true, iff E is satisfiable. Construction is as follows:

E is 3SAT, so it is a conjunction of disjuncts:
$E = D_1 \wedge D_2 \wedge \ldots \wedge D_n$

Program P:
Create atom $falsum$, and for each atom x in E create atoms x and $notx$.

For each x, add two rules of form:
$x \leftarrow \sim notx$ (1)
$notx \leftarrow \sim x$ (2)

and for each disjunct $D_i = (l_1 \vee l_2 \vee l_3)$ add rule
$falsum \leftarrow not(l_1), not(l_2), not(l_3)$ (3)
where $not(l)$ transforms a literal to its "opposite", i.e. $not(x) = notx$, and $not(notx) = x$.

Finally add a rule
$satisfiable \leftarrow \sim falsum$ (4)

Obviously this transformation can be done in polynomial time. Now we show that P has a model where $satisfiable$ is true, if E is satisfiable. Suppose E is satisfiable, let A be the satisfying assignment. So model M, where
$satisfiable \in M$, $\sim falsum \in M$, and for each x: $x \in M \leftrightarrow x = true \in A$, $notx \in M \leftrightarrow x = false \in A$ (which is "semantically equivalent" to satisfying valuation A) is rSM.

Suppose P has a rSM M, where $satisfiable$ is true. For each of literals x,$notx$ exactly one is true (because of rules of form (1), (2)). It must hold that $falsum$ is false (because we have only rule (4) with $satisfiable$ in head), and therefore all rules of form (3) are not applied - their bodies are false. Now suppose an assignment A for E, where each atom x in E is true or false according to if M contains x or $notx$. Each of disjuncts D_i is satisfied in A (because no body from rules (3) is true). So whole E is satisfied.

□

4.2 Complexity of model checking

4.2.1 Preliminaries

It has been shown [3] that minimal model checking is coNP-complete.

Theorem 7. *Given a model M and a program P, it is coNP-complete to decide, if M is a minimal model of P.*

From the definition of stable models it follows that stable model checking can be done in polynomial time. Formally

Theorem 8. *Given a model M and a program P, it the problem if M is a stable model of P is in P.*

Proof. See [9]. □

Now we analyze model checking for Revised Stable Models. We can use the existing implementation described in [12].

Theorem 9. *Given a model M and a program P, it the problem if M is a Revised Stable Model of P is in P.*

Proof. **Model checking**

1. Get model M.
2. Preprocess program P and break OLONs, with respect to M, to get preprocessed program P'.
3. Test if Stable Model of P' is M.

The second step is done by meta-interpreter (see [12] for Prolog implementation) and it takes polynomial time (if we have given M). [1]
Third step can be done in polynomial time (as shown in [9]). □

Now we can prove that Q2 for rSM are solvable in NP:

Lemma 10. *Problem Q2 for revised stable models is in NP.*

Proof. We can write nondeterministic program to compute Q2 in polynomial time.

Nondeterministic program for Q2

1. Guess some model M.
2. Check if M is Revised stable model of P according to Theorem 9.
3. Test if the given literal is in M.

All steps can be done in polynomial time. □

Corollary 11. *Problem Q2 for revised stable models is in NP-complete.*

Proof. Follows from Lemma 6 and Lemma 10. □

[1] For a careful proof, we would have to first proove soundness and completness of the implementation, and then expose it to time analysis. This would be impossible without providing the source code of the implementation and this is out of scope of this work. We can however use an alternative way which does not require an analysis of the source code, but uses a simple transformation of the program. This attitude is described in Chapter 6.

4.3 Cautious reasoning

It has been shown that for SM, cautious reasoning is coNP-complete [11]. Because it's complementary problem to Q2, it holds for rSM, too:

Theorem 12. *Problem Q3 for rSM is coNP-complete.*

4.4 Compilability issues

4.4.1 Motivation

Questions from this section are inspired by [4]. Basic idea is: If we can speed up solving of problems by a pre-compilation, it has a practical value. On the other hand, if the representation of a task in NLP cannot be compiled in this way, we can interpret this as a "very compact" representation, and the intractability is the price we must pay for compactness.
Structure of the problem:

- fixed part (NLP program P)
- variable data (atom, other program, ...)
- question (task)

We can compile the fixed part to anything usable (but of a polynomial size). Compilation may take any time. Now the question is, if solving of problem (with the help of pre-compiled results) takes a polynomial time w.r.t. to size of variable data.

4.4.2 Interesting Questions/Tasks

The fixed part of the problem is a NLP program P. Some of the possibilities what can be the task and what can be the varying part of the problem, are in the following table:

problem	variable data	question/task
0	M	$\mathrm{rSM}(M, P)$?
0b	$\varnothing$	give any model of P
1	a	is there rSM M, $a \in M$?
1b	a	is a in all rSMs?
2	P'	$\exists M : \mathrm{rSM}(M, P) \wedge \mathrm{rSM}(M, P')$
2b	P'	get any such M, $\mathrm{rSM}(M, P) \wedge \mathrm{rSM}(M, P')$
3	P', a	$\exists M : \mathrm{rSM}(M, P) \wedge \mathrm{rSM}(M, P') \wedge a \in M$
3b	P', a	$\forall M : \mathrm{rSM}(M, P) \wedge \mathrm{rSM}(M, P') \wedge a \in M$

Problems **0**, **0b**, **1**, **1b** are similar to problems from first section. Problem **0** can be solved by the algorithm from the proof of Lemma 2 in polynomial time even without a

pre-compilation. Problem **0b** can be solved by implementation [12] without compilation in NP, or can be solved trivially by precompiling some model of P and then can we give this answer in $O(1)$.
Problems **1**, **1b** can be solved by precompiling the answer (true, false) to each atom (in the compilation we can afford computing all models).

Theorem 13. *Problems **2**, **2b**, **3** are NP-complete (and therefore not effectively compilable).*

Proof. Membership can be drawn from Lemma 2. We will show hardness by reduction from 3SAT. We will use P as a some kind of "filter" to get off "uncomfortable models" of program P' . We construct P in a way so it will have 2^n models: For each atom x, we add two rules of the form:
$x \leftarrow \sim notx$
$notx \leftarrow \sim x$

and a rule
$satisfiable \leftarrow \sim falsum$

Now this program has 2^n models, independent combinations of either x or $notx$, and atom $satisfiable$ belongs to all of its models. We can now construct P' in way similar to one in the proof of Lemma 1, and again reduce 3SAT to existence of model (problems **2**, **2b**) or brave reasoning (problem **3**). Note that way of pre-compilation is not relevant. □

Theorem 14. *Problem **3b** is coNP-complete.*

Proof. Problem **3b** is the complement of **3**. □

Chapter 5

Another complexity results

During the work on this theme, we have tried several ways how to describe sustainability. Although those ways have shown themselves unusable, some interesting results remained. I wish to present here such a result. To the the best of my knowledge, it has not appeared before in the literature.

5.1 Computing intersection of all Γ^i

Suppose we want compute the result of $\bigcap_{0\leq i<\omega} \Gamma^i_P(M)$. Direct computation of this intersection is too slow. After finite number of iterations, we get a loop ($\Gamma^i_P(M) = \Gamma^j_P(M)$ for some i, j). However, the the loop can occur too late (if program has OLONs), as is shown by following example.

Example 15. The program:

$a_{11} \leftarrow \sim a_{12} \quad a_{12} \leftarrow \sim a_{11}$

$a_{21} \leftarrow \sim a_{22} \quad a_{22} \leftarrow \sim a_{23} \quad a_{23} \leftarrow \sim a_{21}$

$\vdots$

$a_{i1} \leftarrow \sim a_{i2} \quad a_{i2} \leftarrow \sim a_{i3} \quad \cdots \quad a_{ip_i} \leftarrow \sim a_{i1}$

$\vdots$

$a_{n1} \leftarrow \sim a_{n2} \quad a_{n2} \leftarrow \sim a_{n3} \quad \cdots \quad a_{np_n} \leftarrow \sim a_{n1}$

where p_i is i-th prime, has size of $\sum_{0\leq i\leq n} p_i$, but loop occurs first in $\prod_{0\leq i\leq n} p_i$.

Now we focus on some useful properties of Γ operator (from [1], [6]):

Theorem 16 (Antimonotonicity of Γ)**.** $A \subseteq B \Rightarrow \Gamma(A) \supseteq \Gamma(B)$

Proof. If $A \subseteq B$, then $P/A \supseteq P/B$, and because P/A and P/B are definite programs, their least models must satisfy the same condition, so $M(P/A) \supseteq M(P/B)$. $\square$

Theorem 17 (Monotonicity of Γ^2)**.** $A \subseteq B \Rightarrow \Gamma^2(A) \subseteq \Gamma^2(B)$

Proof. Γ^2 is Γ applied twice. Using the previous theorem we get $A \subseteq B \Rightarrow \Gamma(A) \supseteq \Gamma(B) \Rightarrow \Gamma(\Gamma(A)) \subseteq \Gamma(\Gamma(B))$. □

Theorem 18 (Special property of Γ for models)**.** [1] *Let M be a model. Then $\Gamma(M) \subseteq M$.*

Proof. Theorem 3.1 in [6]. □

We see that in the sequence of iterations $M \supseteq \Gamma(M) \subseteq \Gamma^2(M) \supseteq \Gamma^3(M) \ldots$, we can leave out computing of every even iteration, and instead compute only $I = \bigcap_{i \text{ odd}} \Gamma^i(M) = \bigcap_{0 \leq i < \omega} (\Gamma^2)^i(\Gamma(M))$. Power sets of set of atoms forms a lattice, so we could use following theorem to end computation in polynomial time.

Definition 19 (Branch)**.** Let L be a lattice. Two elements $a, b \in L$ are said to be in one branch, if they are comparable ($a \leq b \vee a \geq b$). The branch is set of the elements which are all comparable to each other (and none element can be added to them). The length of the branch is the number of elements in it. Branch is said to go through an element, if that element belongs to that branch.

Definition 20 (Maximum branching)**.** Maximum number of branches going through a single element is maximum branching of the lattice.

Theorem 21 (Intersection of iterations of a monotonic operator in a lattice)**.** [2] *Let $L(\bigvee, \bigwedge, \preceq, \perp, \top)$ be a lattice, with maximum branching n and maximum branch length m, $\mathcal{F} : L \to L$ be a monotonic operator, and $M \in L$ be an element of the lattice. Then $I_{\mathcal{F}}(M) = \bigwedge_{0 \leq i < \omega} \mathcal{F}^i(M)$ can be computed in $2 \cdot n \cdot m$ steps (evaluations of $\mathcal{F}$) and holds $I_{\mathcal{F}}(M) = \bigwedge_{0 \leq i \leq 2mn} \mathcal{F}^i(M)$.*

Proof. For convenience denote $\mathcal{F}^i(M) = M_i$ (and $M = M_0$), and $I_i = \bigwedge_{0 \leq j \leq i} M_j$. Notation $A \not\approx B$ means A, B are incomparable (i.e. holds none of $A \preceq B$, $A \succ B$).

Suppose we are computing iterations M_i and intermediate results I_i iteratively. We start from $I_0 = M_0 = M$. In each next step (i.e. in computing $(i+1)$-th iteration of $\mathcal{F}$), seven cases can occur (using case discrimination):

a) $M_{i+1} = M_i$: We can stop immediately, with $I_{\mathcal{F}}(M) = I_i$.

[1] This theorem is not absolutely necessary now. We could use Theorem 21 (page 20) to compute intersection of odd and intersection of even iterations of Γ separately See footnote **6** on page 22.

[2] I made this theorem more general. It would be sufficient to assume a power set lattice.

b) $M_{i+1} \succ M_i$: From the monotonicity of $\mathcal{F}$ we have $M_{i+1} = \mathcal{F}(M_i) \preceq \mathcal{F}(M_{i+1}) = M_{i+2}$, and by induction we obtain $\forall j, j \geq i : M_j \preceq M_{j+1}$. An increasing sequence of $M_i \prec M_{i+1} \prec M_{i+2} \prec \ldots$ cannot be longer than m, what means that in at most m steps we get a fixpoint of $\mathcal{F}$, $M_j = M_{j+1}$ for some $j \leq m + i$, and then $I_{\mathcal{F}}(M) = I_j$.

c) $M_{i+1} \prec M_i$: Similarly as above we obtain a decreasing sequence $M_i \succ M_{i+1} \succ M_{i+2} \succ \ldots$, so we can end in at most m steps.

d) $M_{i+1} \not\approx M_i$ and $M_{i+1} \bigwedge I_i \prec I_i$: This means that intermediate result I_i has decreased (to $I_{i+1} \prec I_i$). But a sequence of intermediate results can decrease only at most $|I_i| \leq m$ times. (and then reach $\perp$ – in this case we can immediately end computation). So this situation can occur at most m times in the whole computation.

e) $M_{i+1} \not\approx M_i$ and $M_{i+1} \bigwedge I_i = I_i$ and[3] $\forall j, k,\ i - r \leq j < k \leq i + 1 : M_j \not\approx M_k$, where r is greatest such that in the last r steps no one from cases a, b, c, d occurred: The condition $M_{i+1} \bigwedge I_i = I_i$ means that $M_{i+1} \succeq I_i$, so M_{i+1} is in the same branch as I_i. The last condition means none of elements M_j, M_k are in the same branch. But there are at most n different branches going through I_i, so r can reach at most $n-2$, and this step can successively repeat at most $n - 1$ times.

f) $M_{i+1} \not\approx M_i$ and $M_{i+1} \bigwedge I_i = I_i$ and[4] $\exists j,\ i-r \leq j < i : M_j \preceq M_i$, r like in e): From the monotonicity of $\mathcal{F}$ we have $M_{j+l} \preceq M_{i+l} \Rightarrow M_{j+l+1} = \mathcal{F}(M_{j+l}) \preceq \mathcal{F}(M_{i+l}) = M_{i+l+1}$, what we can extend by induction on l to $\forall l \in \mathbb{N}_0 : M_{j+l} \preceq M_{i+l}$. Now we can substitute $l = k{\cdot}(i-j)$ to get $\forall k \in \mathbb{N}_0 : M_{j+k\cdot(i-j)} \preceq M_{i+k\cdot(i-j)} = M_{j+(k+1)(i-j)}$. That leads us to the non-decreasing sequence

$$M_j \preceq M_{j+(i-j)} \preceq M_{j+2\cdot(i-j)} \preceq \ldots \preceq M_{j+k\cdot(i-j)} \preceq M_{j+(k+1)\cdot(i-j)} \preceq \ldots$$

This sequence can have at most m different elements, so in at most $m{\cdot}(i-j) \leq m{\cdot}n$ steps we reach $M_r = M_s$ for some $r < s < i + m{\cdot}n$. That is a loop and we can stop the computation with $I_{\mathcal{F}}(M) = I_s$.

g) $M_{i+1} \not\approx M_i$ and $M_{i+1} \bigwedge\ I_i = I_i$ and $\nexists j,\ i - r \leq j < i : M_j \preceq M_i$ and[5] $\exists j,\ i - r \leq j < i : M_j \succeq M_i$, r like in the case e: Similar as in f).

In the cases a, b, c, f, g we can immediately limit number of steps needed to compute the result. It means when the computation in some step i reaches one of these cases, number of steps could be limited by $i + n{\cdot}m$. Other two cases (d, e) can alternate at most $n{\cdot}m$ steps, then the computation will reach the end (in d when $I_i = \perp$) or one of a, b, c, f, g occurs. Therefore the whole computing takes at most $2{\cdot}m{\cdot}n$ steps. □

[3] Following condition implies first condition ($M_{i+1} \not\approx M_i$), but for the purpose of case discrimination is the first condition presented separately.

[4] First two conditions are presented only for the purpose of case discrimination and are not used.

[5] Again case discrimination, interesting is only the last condition.

Remark 22. Because of the nature of lattices, it would be possible to compute the join of iterations, too: $J_{\mathcal{F}}(M) = \bigvee_{0\leq i<\omega} \mathcal{F}^i(M) = \bigvee_{0\leq i\leq 2mn} \mathcal{F}^i(M)$. Proof is similar.

Corollary 23. The result of $\bigcap_{0\leq i<\omega} \Gamma^i_P(M)$ can be computed in polynomial time w. r. to size of P.

Proof. Let n be number of literals (and that is equal to both maximal length of branch and maximal branching in lattice of interpretations). Denote operator $\mathcal{F} = \Gamma^2$. Evaluation of Γ can be done in linear time, so evaluation of $\mathcal{F}$ is linear, too. Computing $I_{\mathcal{F}}(\Gamma(M))$ according to previous theorem is $2n^2$ applications of $\mathcal{F}$ and therefore in time $2n^3$. [6] $\square$

[6] Here we can avoid using the trick of Theorem 18 (page 20) and compute directly $I = I_{\mathcal{F}}(M) \cap I_{\mathcal{F}}(\Gamma(M))$. This will work for some M which does not have to be a model, too.

Chapter 6

Introducing *reductio ad absurdum* by transformations

6.1 Criticism of Revised Stable Models

The sustainability notion is the main reason of incomprehensibility of RSM. It is a complex concept which is not easily understandable. Next reason is the semantics. Although, we can eventually use a new semantics, programmers are used to the old. Therefore, it will take a long/difficult time to propagate the idea among others. I suggest to overcome these limitations by an alternative way: to use a transformation on top of existing SM semantics. This approach is presented in this chapter.

6.2 Removing OLONs

As Pereira and Pinto [13] recognized, the main problem of introducing *reductio ad absurdum* into SM, are OLONs (in general programs), and ICONs (in infinite/functional programs). They chose a high-level approach to resolving this issue by a new semantics - RSM. In this chapter we suggest a transformation inspired by thoughts of Pinto and his implementation of RSM, although in a generalized way: we do not limit ourselves to a particular definition of *reductio ad absurdum*, we can use RSM/MM semantics, or whatever of our choice. We will describe how to remove OLONs from the program and so enable use of stable models to gain a model. In practical use, we can have an ASP-solver to do it.

6.2.1 Transformation

1. Identify OLON

2. Compute the model of the subprogram (of the OLON) according to chosen way of computing reductio ad absurdum

3. Replace OLON by a new subprogram, which has the same semantics

4. Feed the whole program into Stable Models

5. If you introduced auxiliary atoms, filter them out

For the second step, we can use RSM or minimal models, or an another semantics of our choice, which will resolve OLON in the way we like. In the third step, we can use any way how to construct a program. For RSM, we can use this transformation:

Definition 24 (Breaking OLONs in Revised Stable Models)**.** Let $x_1, \ldots, x_n$ be the atoms in the OLON. Then together with rules of the OLON, add extra rules:

$$
\begin{array}{l}
x_1 \leftarrow \sim x_2, \ldots, \sim x_n \\
x_2 \leftarrow \sim x_1, \sim x_3, \ldots, \sim x_n \\
\vdots \\
x_i \leftarrow \sim x_1, \ldots, \sim x_{i-1}, \sim x_{i+1}, \sim x_n \\
\vdots \\
x_n \leftarrow \sim x_1, \ldots, \sim x_{n-1}
\end{array}
$$

We can see how the extra rules helps to "boot" the derivation of *reductio ad absurdum*.

For minimal models, or generally for any semantics, we can construct the program in a following way:
Create sufficiently enough new atoms $x_1, \ldots, x_n$ and create program which creates models by binary switching each atom on/off, i.e. 2^n models. We can think of each model as a representant for a binary number with length n, with its binary digit i being 1 if $x_i \in M$ or 0 if not.
Now we can use each of those models to trigger a set of rules. Each set will introduce a model from the set of models of the OLON computed in step 2. If we have not used all of 2^n models (representatnts for numbers), we will invalidate them by using an integrity constraint to avoid an empty stable model.

Definition 25 (Breaking OLONs according to Minimal models semantics)**.** Let $M_0, \ldots, M_m$ be minimal models of chosen OLON. Let $n = \lceil log_2(m+1) \rceil - 1$. Replace OLON rules by

$$
\begin{array}{l}
x_0 \leftarrow \sim x_0 \\
x_0 \leftarrow \sim nx_0
\end{array}
$$

$\vdots$

$x_n \leftarrow \sim x_n$
$x_n \leftarrow \sim nx_n$

Let us use rule in form $M \leftarrow x$ to denote the set of rules $\{y \leftarrow x | y \in M\}$. Now add rules

$M_0 \leftarrow \sim x_0, \sim x_1, \ldots, \sim x_n$
$M_1 \leftarrow x_0, \sim x_1, \ldots, \sim x_n$
$\vdots$
$M_i \leftarrow \{x_k \mid k\text{-th bit is set in binary representation of } i\}, \{\sim x_k \mid k\text{-th bit is not set in } i\}$
$\vdots$
$M_m \leftarrow x_0, x_1, \ldots, x_n$

6.3 Examples

original	RSM transformation	MM transformation
$a \leftarrow \sim b$	$a \leftarrow \sim b$	$x_0 \leftarrow \sim nx_0$
$b \leftarrow \sim c$	$b \leftarrow \sim c$	$nx_0 \leftarrow \sim x_0$
$c \leftarrow \sim a$	$c \leftarrow \sim a$	$x_1 \leftarrow \sim nx_1$
		$x_1 \leftarrow \sim nx_1$
	$a \leftarrow \sim b, \sim c$	$a \leftarrow \sim x_0, \sim x_1$
	$b \leftarrow \sim a, \sim c$	$b \leftarrow \sim x_0, \sim x_1$
	$c \leftarrow \sim a, \sim b$	$b \leftarrow x_0, \sim x_1$
		$c \leftarrow x_0, \sim x_1$
		$a \leftarrow \sim x_0, x_1$
		$c \leftarrow \sim x_0, x_1$
		$falsum \leftarrow x_0, x_1, \sim falsum$

The original program has no stable model. The program in the second column has stable models $\{a, b\}, \{b, c\}, \{a, c\}$ which corresponds to the first program's revised stable models. The program in the third column has stable models $\{a, b, nx_0, nx_1\}, \{b, c, x_0, nx_1\}, \{a, c, nx_0, x_1\}$. If we filter out the auxiliary atoms, it corresponds to the previous program (because in this case, minimal models and revised stable models are the same).

original	RSM transformation	MM transformation
$a \leftarrow \sim b, \sim d$	$a \leftarrow \sim b, \sim d$	$x_0 \leftarrow \sim nx_0$
$b \leftarrow \sim a$	$b \leftarrow \sim a$	$nx_0 \leftarrow \sim x_0$
$b \leftarrow a, c$	$b \leftarrow a, c$	$x_1 \leftarrow \sim nx_1$
$c \leftarrow \sim b, \sim c, \sim d$	$c \leftarrow \sim b, \sim c, \sim d$	$x_1 \leftarrow \sim nx_1$
$d \leftarrow \sim a, b, \sim d$	$d \leftarrow \sim a, b, \sim d$	$b \leftarrow \sim x_0, \sim x_1$
		$d \leftarrow \sim x_0, \sim x_1$
	$a \leftarrow \sim b, \sim c, \sim d$	$a \leftarrow x_0, \sim x_1$
	$b \leftarrow \sim a, \sim c, \sim d$	$b \leftarrow x_0, \sim x_1$
	$c \leftarrow \sim a, \sim b, \sim d$	$a \leftarrow \sim x_0, x_1$
	$d \leftarrow \sim a, \sim b, \sim c$	$d \leftarrow \sim x_0, x_1$
		$falsum \leftarrow x_0, x_1, \sim falsum$

In this example, the original program has no stable model again. It has only Revised Stable Models, which are the same as stable models of the second program and that are: $\{a, b\}, \{b, d\}$. The third program has three stable models, $\{b, d, nx_0, nx_1\}, \{a, b, x_0, nx_1\}, \{a, d, nx_0, x_1\}$, what after filtering corresponds to minimal models of the first program, $\{b, d\}, \{a, b\}, \{a, d\}$. The semantics of the Revised Stable Models and Minimal Models is not the same here.

6.4 Complexity of the transformation and computing semantics of the transformed program

6.4.1 Complexity of the Revised Stable Model transformation

We see that transformation can be done easily by adding rules for each of involved atoms. This can obviously be done in polynomial time. Checking of stable models can be done in polynomial time, too.

6.4.2 Complexity of the general transformation

Transformation can be done easily by adding rules and this can again be done in polynomial time. However, we have to compute each of the desired model. For minimal models this is an exponential time complexity. Checking of stable models can is then in polynomial time.

Chapter 7

Conclusions

7.1 Future work

7.1.1 Revised Stable Models with explicit negations

Maybe we can use the transformation proposed in the previous chapter in Definition 24 to introduce *reductio ad absurdum* into programs with explicit negations. We do not see a Principal problem to carry out this. However, no research in this direction was made and we consider this as an open problem out of scope of this work.

7.1.2 Investigate practical issues connected with introducing RSM

It will be nice to implement an user friendly module to compute Revised Stable Model semantics, like `smodels` package for Stable Model semantics. This will enable programmers to efficiently use *reductio ad absurdum* reasoning.

7.2 Others' work

A research concerning use of Revised Stable Model in dynamic logic programming is being done in CENTRIA[1]. Another research by Luís Rodrigues Soares is to provide a fixpoint definition of Revised Stable Models in so-called Revised Well-founded Semantics.

7.3 Epilogue

We have led the reader through the Revised Stable model semantics and its complexity. We presented examples, ideas, theorems and proofs of the topic, concentrating on the

[1] Centro de Inteligência Artificial – Centre for Artificial Intelligence, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Portugal
http://centria.di.fct.unl.pt

complexity. We provided a useful by-product of the research in form of the algebraic theorem. For the convenience of the reader, we tried to visualize concepts in a graphical way.
Complexity of Revised Stable models is acceptable, because it is no worse as that of Stable Models. Together with *reductio ad absurdum* reasoning and other good properties it can overweight the more complex definition of the semantics and become a perspective for future logic programmers.

GNU Free Documentation License

Version 1.2, November 2002
Copyright © 2000,2001,2002 Free Software Foundation, Inc.

51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the

Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software

Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ... Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

List of Figures

Bibliography

[1] Chitta R. Baral and V. S. Subrahmanian. Stable and extension class theory for logic programs and default logics. *J. Autom. Reason.*, 8(3):345–366, 1992.

[2] Encyclopædia Britannica. reductio ad absurdum. from Encyclopædia Britannica Online: `http://www.britannica.com/eb/article-9062992`.

[3] Marco Cadoli. The complexity of model checking for circumscriptive formulae. *Inf. Process. Lett.*, 44(3):113–118, 1992.

[4] Marco Cadoli, Francesco M. Donini, and Marco Schaerf. Is intractability of nonmonotonic reasoning a real drawback? *Artif. Intell.*, 88(1-2):215–251, 1996.

[5] Wikipedia contributors. Reductio ad absurdum — Wikipedia, The Free Encyclopedia, 2007. [Online; accessed 1-May-2007] `http://en.wikipedia.org/w/index.php?title=Reductio_ad_absu%rdum&oldid=126055014`.

[6] Stefania Costantini. Contributions to the stable model semantics of logic programs with negation. *Theoretical Computer Science*, 149(2):231–255, 1995.

[7] Marc Denecker, Victor W. Marek, and Mirosław Truszczyński. Uniform semantic treatment of default and autoepistemic logics. *Artif. Intell.*, 143(1):79–122, 2003.

[8] John Ellson – AT&T Research et al. Graphviz - graph visualization software. `http://www.graphviz.org`.

[9] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Procs. of ICLP-88*, pages 1070–1080. International Conference on Logic Programming 88, 1988.

[10] Victor W. Marek and Mirosław Truszczyński. Autoepistemic logic. *Journal of the ACM*, 38(3):588–619, 1991.

[11] Victor W. Marek and Mirosław Truszczyński. Computing intersection of autoepistemic expansions. In Anil Nerode, Victor W. Marek, and V. S. Subrahmanian, editors, *Logic Programming and Non-Monotonic Reasoning, Proceedings of the first International Workshop*, pages 37–50. MIT Press, 1991.

[12] Luís Moniz Pereira and Alexandre Miguel Pinto. Implementing the revised stable models – an asp- based approach. 2005. Submitted to Annals of Mathematics and Artificial Intelligence.

[13] Luís Moniz Pereira and Alexandre Miguel Pinto. Revised stable models - a semantics for logic programs. In G. Dias C. Bento, A. Cardoso, editor, *Procs. 12th Portuguese Intl. Conf. on Artificial Intelligence (EPIA'05)*, pages 29–42, Covilhã, Portugal, December 2005. Springer.

[14] A. van Gelder. The alternating fixpoint of logic programs with negation. *Journal of computer and system sciences*, 47:185–221, 1993.

Abstract (english)

The theme of the work is to analyze the complexity of Revised Stable Models. Revised Stable models are a new semantics in logical programming, which brings a possibility of the so-called reductio ad absurdum reasoning, which was not possible in the stable models semantics. At the beginning, there are presented the preliminaries and a motivation for the semantics, its definition and an intuitive explanation and visualization. Next come the analyze of the complexity and a free algebraic follow-up which describes the complexity of computing intersection of iteration of a monotonic operator in the lattice. An alternative view of the semantics is drawn, which offers a possibility to introduce the reductio ad absurdum into stable models by using a transformation of the program. The work concludes with an outline of possible future research themes.

Keywords: logic programming, semantics, Stable Models, reductio ad absurdum, complexity, lattice

Abstrakt (slovenský)

Práca sa zaoberá skúmaním zložitosti revidovaných stabilných modelov. Revidované stabilné modely sú nová sémantika v logickom programovaní, ktorá prináša možnosť odvodzovania za pomoci argumentácie cez tzv. reductio ad absurdum, ktorú sémantika stabilných modelov neumožňuje. Na začiatok sú vysvetlené východiská a motivácia pre zavedenie sémantiky, jej definícia a intuitívne objasnenie a znázornenie. Nasleduje analýza zložitosti, na ktorú voľne nadväzuje algebraický výsledok, popisujúci zložitosť počítania prieniku iterácií monotónneho operátora na zväzoch. Je načrtnutý alternatívny pohľad na sémantiku revidovaných stabilných modelov, kde sa ponúka možnosť zavedenia reductio ad absurdum do stabilných modelov cez transformácie programu. Prácu uzatvára prehľad možného pokračovania výskumu v tejto oblasti.

Kľúčové slová: logické programovanie, sémantika, stabilné modely, reductio ad absurdum, zložitosť, zväz