Every normal logic program has a 2-valued Minimal Hypotheses semantics

Alexandre Miguel Pinto and Luís Moniz Pereira
{amp|lmp}@di.fct.unl.pt

Centro de Inteligência Artificial (CENTRIA), Departamento de Informática
Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa
2829-516 Caparica, Portugal

Abstract. In this paper we explore a unifying approach — that of hypotheses assumption — as a means to provide a semantics for all Normal Logic Programs (NLPs), the Minimal Hypotheses (MH) semantics [1]. This semantics takes a positive hypotheses assumption approach as a means to guarantee the desirable properties of model existence, relevance and cumulativity, and of generalizing the Stable Models semantics in the process. To do so we first introduce the fundamental semantic concept of minimality of assumed positive hypotheses, define the MH semantics, and analyze the semantics' properties and applicability. Indeed, abductive Logic Programming can be conceptually captured by a strategy centered on the assumption of abducibles (or hypotheses). Likewise, the Argumentation perspective of Logic Programs (e.g. [5]) also lends itself to a arguments (or hypotheses) assumption approach. Previous works on Abduction (e.g. [10]) have depicted the atoms of default negated literals in Normal Logic Programs as abducibles, i.e., assumable hypotheses. We take a complementary and more general view than these works to NLP semantics by employing positive hypotheses instead.
Keywords: Hypotheses, Semantics, NLPs, Abduction, Argumentation.

1 Background

Logic Programs have long been used in Knowledge Representation and Reasoning.

Definition 1. ***Normal Logic Program.*** *By an alphabet $\mathcal{A}$ of a language $\mathcal{L}$ we mean (finite or countably infinite) disjoint sets of constants, predicate symbols, and function symbols, with at least one constant. In addition, any alphabet is assumed to contain a countably infinite set of distinguished variable symbols. A term over $\mathcal{A}$ is defined recursively as either a variable, a constant or an expression of the form $f(t_1, \ldots, t_n)$ where f is a function symbol of $\mathcal{A}$, n its arity, and the t_i are terms. An atom over $\mathcal{A}$ is an expression of the form $P(t_1, \ldots, t_n)$ where P is a predicate symbol of $\mathcal{A}$, and the t_i are terms. A literal is either an atom A or its default negation $not\ A$. We dub default literals (or default negated literals — DNLs, for short) those of the form $not\ A$. A term*

[1] This paper is a very condensed summary of some of the main contributions of the PhD Thesis [17] of the first author, supported by FCT-MCTES grant SFRH / BD / 28761 / 2006, and supervised by the second author.

(resp. atom, literal) is said ground if it does not contain variables. The set of all ground terms (resp. atoms) of $\mathcal{A}$ is called the Herbrand universe (resp. base) of $\mathcal{A}$. For short we use $\mathcal{H}$ to denote the Herbrand base of $\mathcal{A}$. A Normal Logic Program (NLP) is a possibly infinite set of rules (with no infinite descending chains of syntactical dependency) of the form

$$H \leftarrow B_1, \ldots, B_n, not\ C_1, \ldots, not\ C_m, \text{ (with } m, n \geq 0 \text{ and finite)}$$

where H, the B_i and the C_j are atoms, and each rule stands for all its ground instances. In conformity with the standard convention, we write rules of the form $H \leftarrow$ also simply as H (known as "facts"). An NLP P is called definite if none of its rules contain default literals. H is the head of the rule r, denoted by $head(r)$, and $body(r)$ denotes the set $\{B_1, \ldots, B_n, not\ C_1, \ldots, not\ C_m\}$ of all the literals in the body of r.

When doing problem modelling with logic programs, rules of the form

$$\bot \leftarrow B_1, \ldots, B_n, not\ C_1, \ldots, not\ C_m, \text{ (with } m, n \geq 0 \text{ and finite)}$$

with a non-empty body are known as a type of Integrity Constraints (ICs), specifically *denials*, and they are normally used to prune out unwanted candidate solutions. We abuse the 'not ' default negation notation applying it to non-empty sets of literals too: we write $not\ S$ to denote $\{not\ s : s \in S\}$, and confound $not\ not\ a \equiv a$. When S is an arbitrary, non-empty set of literals $S = \{B_1, \ldots, B_n, not\ C_1, \ldots, not\ C_m\}$ we use

- S^+ denotes the set $\{B_1, \ldots, B_n\}$ of positive literals in S
- S^- denotes the set $\{not\ C_1, \ldots, not\ C_m\}$ of negative literals in S
- $|S| = S^+ \cup (not\ S^-)$ denotes the $\{B_1, \ldots, B_n, C_1, \ldots, C_m\}$ of atoms of S

As expected, we say a set of literals S is consistent iff $S^+ \cap |S^-| = \emptyset$. We also write $heads(P)$ to denote the set of heads of non-IC rules of a (possibly constrained) program P, i.e., $heads(P) = \{head(r) : r \in P\} \setminus \{\bot\}$, and $facts(P)$ to denote the set of facts of P — $facts(P) = \{head(r) : r \in P \wedge body(r) = \emptyset\}$.

Definition 2. ***Part of body of a rule not in loop.*** *Let P be an NLP and r a rule of P. We write $\overline{body(r)}$ to denote the subset of $body(r)$ whose atoms do not depend on r. Formally, $\overline{body(r)}$ is the largest set of literals such that*

$$\overline{body(r)} \subseteq body(r) \wedge \forall_{a \in |\overline{body(r)}|} \nexists_{r_a \in P} head(r_a) = a \wedge r_a \twoheadleftarrow r$$

where $r_a \twoheadleftarrow r$ means rule r_a depends on rule r, i.e., either $head(r) \in |body(r_a)|$ or there is some other rule $r' \in P$ such that $r_a \twoheadleftarrow r'$ and $head(r) \in |body(r')|$.

Definition 3. ***Layer Supported and Classically supported interpretations.*** *We say an interpretation I of an NLP P is layer (classically) supported iff every atom a of I is layer (classically) supported in I. a is layer (classically) supported in I iff there is some rule r in P with $head(r) = a$ such that $I \models \overline{body(r)}$ ($I \models body(r)$). Likewise, we say the rule r is layer (classically) supported in I iff $I \models \overline{body(r)}$ ($I \models body(r)$).*

Literals in $\overline{body(r)}$ are, by definition, not in loop with r. The notion of layered support requires that all such literals be *true* under I in order for $head(r)$ to be layer supported in I. Hence, if $\overline{body(r)}$ is empty, $head(r)$ is *ipso facto* layer supported.

Proposition 1. ***Classical Support implies Layered Support.*** *Given a NLP P, an interpretation I, and an atom a such that $a \in I$, if a is classically supported in I then a is also layer supported in I.*

Proof. Knowing that, by definition, $\overline{body(r)} \subseteq body(r)$ for every rule r, it follows trivially that a is layer supported in I if a is classically supported in I.

2 Motivation

"Why the need for another 2-valued semantics for NLPs since we already have the Stable Models one?" The question has its merit since the Stable Models (SMs) semantics [7] is exactly what is necessary for so many problem solving issues, but the answer to it is best understood when we ask it the other way around: "Is there any situation where the SMs semantics does not provide all the intended models?" and "Is there any 2-valued generalization of SMs that keeps the intended models it does provide, adds the missing intended ones, and also enjoys the useful properties of guarantee of model existence, relevance, and cumulativity?"

Example 1. **A Joint Vacation Problem — Merging Logic Programs.** Three friends are planning a joint vacation. First friend says "If we don't go to the mountains, then we should go to the beach". The second friend says "If we don't go to travelling, then we should go to the mountains". The third friend says "If we don't go to the beach, then we should go travelling". We code this information as the following NLP:

$$\begin{aligned} beach &\leftarrow not\ mountain \\ mountain &\leftarrow not\ travel \\ travel &\leftarrow not\ beach \end{aligned}$$

Each of these individual consistent rules come from a different friend. According to the SMs, each friend had a "solution" (a SM) for his own rule, but when we put the three rules together, because they form an odd loop over negation, the resulting merged logic program has no SM. If we assume $beach$ is *true* then we cannot conclude $travel$ and therefore we conclude $mountain$ is also *true* — this gives rise to the $\{beach, mountain, not\ travel\}$ joint and multi-place vacation solution. The other (symmetric) two are $\{mountain, not\ beach, travel\}$ and $\{travel, not\ mountain, beach\}$. This example too shows the importance of having a 2-valued semantics guaranteeing model existence, in this case for the sake of arbitrary merging of logic programs (and for the sake of existence of a joint vacation for these three friends).

Increased Declarativity. An IC is a rule whose head is $\bot$, and although such syntactical definition of IC is generally accepted as standard, the SM semantics can employ odd loops over negation, such as the $a \leftarrow not\ a, X$ to act as ICs, thereby mixing and unnecessarily confounding two distinct Knowledge Representation issues: the one of IC use, and the one of assigning semantics to loops. For the sake of declarativity, rules with $\bot$ head should be the only way to write ICs in a LP: no rule, or combination of rules, with head different from $\bot$ should possibly act as IC(s) under any given semantics.

Argumentation From an argumentation perspective, the author of [5], states:

> *"Stable extensions do not capture the intuitive semantics of every meaningful argumentation system."*

where the "stable extensions" have a one-to-one correspondence to the Stable Models ([5]), and also

> *"Let P be a knowledge base represented either as a logic program, or as a nonmonotonic theory or as an argumentation framework. Then there is not necessarily a "bug" in P if P has no stable semantics.*
>
> *This theorem defeats an often held opinion in the logic programming and nonmonotonic reasoning community that if a logic program or a nonmonotonic theory has no stable semantics then there is something "wrong" in it."*

Thus, a criterion different from the *stability* one must be used in order to effectively model every argumentation framework adequately.

Arbitrary Updates and/or Merges One of the main goals behind the conception of non-monotonic logics was the ability to deal with the changing, evolving, updating of knowledge. There are scenarios where it is possible and useful to combine several Knowledge Bases (possibly from different authors or sources) into a single one, and/or to update a given KB with new knowledge. Assuming the KBs are coded as IC-free NLPs, as well as the updates, the resulting KB is also an IC-free NLP. In such a case, the resulting (merged and/or updated) KB should always have a semantics. This should be true particularly in the case of NLPs where no negations are allowed in the heads of rules. In this case no contradictions can arise because there are no conflicting rule heads. The lack of such guarantee when the underlying semantics used is the Stable Models, for example, compromises the possibility of arbitrarily updating and/or merging KBs (coded as IC-free NLPs). In the case of self-updating programs, the desirable "liveness" property is put into question, even without outside intervention.

These motivational issues raise the questions "Which should be the 2-valued models of an NLP when it has no Stable Models?", "How do these relate to SMs?", "Is there a uniform approach to characterize both such models and the SMs?", and "Is there any 2-valued generalization of the SMs that encompasses the intuitive semantics of *every* logic program?". Answering such questions is a paramount motivation and thrust in this paper.

2.1 Intuitively Desired Semantics

It is commonly accepted that the non-stratification of the default not is the fundamental ingredient which allows for the possibility of existence of several models for a program. The non-stratified DNLs (i.e., in a loop) of a program can thus be seen as non-deterministically assumable choices. The rules in the program, as well as the particular semantics we wish to assign them, is what constrains which sets of those choices we take as acceptable.

2.2 Desirable Formal Properties

Only ICs (rules with $\bot$ head) should "endanger" model existence in a logic program. Therefore, a semantics for NLPs with no ICs should guarantee model existence — which, e.g., does not occur with SM semantics. Relevance is also a useful property since it allows the development of top-down query-driven proof-procedures that allow for the sound and complete search for answers to a user's query. This is useful in the sense that in order to find an answer to a query only the relevant part of the program must be considered, whereas with a non-relevant semantics the whole program must be considered, with corresponding performance disadvantage compared to a relevant semantics.

Definition 4. ***Relevant part of*** P ***for atom*** a**.** *The relevant part of NLP* P *for atom* a *is* $Rel_P(a) = \{r_a \in P : head(r_a) = a\} \cup \{r \in P : \exists_{r_a \in P \wedge head(r_a)=a} r_a \twoheadleftarrow r\}$

Definition 5. ***Relevance (adapted from [3]).*** *A semantics* Sem *for logic programs is said Relevant iff for every program* P

$$\forall_{a \in \mathcal{H}_P}(\forall_{M \in Models_{Sem}(P)} a \in M) \Leftrightarrow (\forall_{M_a \in Models_{Sem}(Rel_P(a))} a \in M_a)$$

Moreover, cumulativity also plays a role in performance enhancement in the sense that only a semantics enjoying this property can take advantage of storing intermediate lemmas to speed up future computations.

Definition 6. ***Cumulativity (adapted from [4]).*** *Let* P *be an NLP, and* a, b *two atoms of* $\mathcal{H}_P$*. A semantics* Sem *is Cumulative iff the semantics of* P *remains unchanged when any atom* true *in the semantics is added to* P *as a fact:*

$$\forall_{a,b \in \mathcal{H}_P} \big((\forall_{M \in Models_{Sem}(P)} a \in M) \Rightarrow$$

$$(\forall_{M \in Models_{Sem}(P)} b \in M \Leftrightarrow \forall_{M_a \in Models_{Sem}(P \cup \{a\})} b \in M_a)\big)$$

Finally, each individual SM of a program, by being minimal and classically supported, should be accepted as a model according to every 2-valued semantics, and hence every 2-valued semantics should be a model conservative extension of Stable Models.

3 Syntactic Operations

It is commonly accepted that definite LPs (i.e., without default negation) have only one 2-valued model — its *least model* which coincides with the well-founded model. This is also the case for locally stratified LPs. In such cases we can use a deterministic operator performing syntactic transformations on a program to obtain that model. In [2] the author defined the program Remainder operator (denoted by $\widehat{P}$) for calculating the Well-Founded Model, which coincides with the unique perfect model for locally stratified LPs. The Remainder can thus be seen as a generalization for NLPs of the T operator, the latter applicable only to the subclass of definite LPs. We recap here the definitions necessary for the Remainder operator because we will use it in the definition of our Minimal Hypotheses semantics. The intuitive gist of MH semantics (formally

defined in section 4) is as follows: an interpretation M_H is a MH model of program P iff there is some minimal set of hypotheses H such that the truth-values of all atoms of P become determined assuming the atoms in H as *true*. We resort to the program Remainder operator as a deterministic (and efficient, i.e., computable in polynomial time) means to find out if the truth-values of all literals became determined or not — we will see below how the Remainder can be used to find this out.

3.1 Program Remainder operator

For self-containment, we include here the definitions of [2] upon which the Remainder operator relies, and adapt them where convenient to better match the syntactic conventions used throughout this paper.

Definition 7. ***Program transformation (def. 4.2 of [2]).*** *A* program transformation *is a relation $\mapsto$ between ground logic programs. A semantics S allows a transformation $\mapsto$ iff $Models_S(P_1) = Models_S(P_2)$ for all P_1 and P_2 with $P_1 \mapsto P_2$. We write $\mapsto^*$ to denote the fixed point of the $\mapsto$ operation, i.e., $P \mapsto^* P'$ where $\nexists_{P'' \neq P'} P' \mapsto P''$. It follows that $P \mapsto^* P' \Rightarrow P' \mapsto P'$.*

Definition 8. ***Positive reduction (def. 4.6 of [2]).*** *Let P_1 and P_2 be ground programs. Program P_2 results from P_1 by* positive reduction *$(P_1 \mapsto_P P_2)$ iff there is a rule $r \in P_1$ and a negative literal $not\ b \in body(r)$ such that $b \notin heads(P_1)$, i.e., there is no rule for b in P_1, and $P_2 = (P_1 \setminus \{r\}) \cup \{head(r) \leftarrow (body(r) \setminus \{not\ b\})\}$.*

Definition 9. ***Negative reduction (def. 4.7 of [2]).*** *Let P_1 and P_2 be ground programs. Program P_2 results from P_1 by* negative reduction *$(P_1 \mapsto_N P_2)$ iff there is a rule $r \in P_1$ and a negative literal $not\ b \in body(r)$ such that $b \in facts(P_1)$, i.e., b appears as a fact in P_1, and $P_2 = P_1 \setminus \{r\}$.*

Negative reduction is consistent with classical support, but not with the layered one. Therefore, we introduce now a layered version of the negative reduction operation.

Definition 10. ***Layered negative reduction.*** *Let P_1 and P_2 be ground programs. Program P_2 results from P_1 by* layered negative reduction *$(P_1 \mapsto_{LN} P_2)$ iff there is a rule $r \in P_1$ and a negative literal $not\ b \in \overline{body(r)}$ such that $b \in facts(P_1)$, i.e., b appears as a fact in P_1, and $P_2 = P_1 \setminus \{r\}$.*

The Strongly Connected Components (SCCs) of rules of a program can be calculated in polynomial time [18]. Once the SCCs of rules have beed identified, the $\overline{body(r)}$ subset of $body(r)$, for each rule r, is identifiable in linear time — one needs to check just once for each literal in $body(r)$ if it is also in $\overline{body(r)}$. Therefore, these polynomial time complexity operations are all the added complexity Layered negative reduction adds over regular Negative reduction.

Definition 11. ***Success (def. 5.2 of [2]).*** *Let P_1 and P_2 be ground programs. Program P_2 results from P_1 by* success *$(P_1 \mapsto_S P_2)$ iff there are a rule $r \in P_1$ and a fact $b \in facts(P_1)$ such that $b \in body(r)$, and $P_2 = (P_1 \setminus \{r\}) \cup \{head(r) \leftarrow (body(r) \setminus \{b\})\}$.*

Definition 12. ***Failure (def. 5.3 of [2]).*** *Let P_1 and P_2 be ground programs. Program P_2 results from P_1 by* failure $(P_1 \mapsto_F P_2)$ *iff there are a rule $r \in P_1$ and a positive literal $b \in body(r)$ such that $b \notin heads(P_1)$, i.e., there are no rules for b in P_1, and $P_2 = P_1 \setminus \{r\}$.*

Definition 13. ***Loop detection (def. 5.10 of [2]).*** *Let P_1 and P_2 be ground programs. Program P_2 results from P_1 by* loop detection $(P_1 \mapsto_L P_2)$ *iff there is a set $\mathcal{A}$ of ground atoms such that*

1. *for each rule $r \in P_1$, if $head(r) \in \mathcal{A}$, then $body(r) \cap \mathcal{A} \neq \emptyset$,*
2. *$P_2 := \{r \in P_1 | body(r) \cap \mathcal{A} = \emptyset\}$,*
3. *$P_1 \neq P_2$.*

We are not entering here into the details of the *loop detection* step, but just taking note that 1) such a set $\mathcal{A}$ corresponds to an unfounded set (cf. [6]); 2) loop detection is computationally equivalent to finding the SCCs [18], and is known to be of polynomial time complexity; and 3) the atoms in the unfounded set $\mathcal{A}$ have all their corresponding rules involved in SCCs where all heads of rules in loop appear positive in the bodies of the rules in loop.

Definition 14. ***Reduction (def. 5.15 of [2]).***
Let $\mapsto_X$ denote the rewriting system: $\mapsto_X := \mapsto_P \cup \mapsto_N \cup \mapsto_S \cup \mapsto_F \cup \mapsto_L$.

Definition 15. ***Layered reduction.***
Let $\mapsto_{LX}$ denote the rewriting system: $\mapsto_{LX} := \mapsto_P \cup \mapsto_{LN} \cup \mapsto_S \cup \mapsto_F \cup \mapsto_L$.

Definition 16. ***Remainder (def. 5.17 of [2]).*** *Let P be a program. Let $\widehat{P}$ satisfy $ground(P) \mapsto_X^* \widehat{P}$. Then $\widehat{P}$ is called the* remainder *of P, and is guaranteed to exist and to be unique to P. Moreover, the calculus of $\mapsto_X^*$ is known to be of polynomial time complexity [2]. When convenient, we write $Rem(P)$ instead of $\widehat{P}$.*

An important result from [2] is that the Well-Founded Model (WFM — [6]) of P is such that $WFM^+(P) = facts(\widehat{P})$, $WFM^{+u} = heads(\widehat{P})$, and $WFM^-(P) = \mathcal{H}_P \setminus WFM^{+u}(P)$, where $WFM^+(P)$ denotes the set of atoms of P *true* in the WFM, $WFM^{+u}(P)$ denotes the set of atoms of P *true* or *undefined* in the WFM, and $WFM^-(P)$ denotes the set of atoms of P *false* in the WFM.

Definition 17. ***Layered Remainder.*** *Let P be a program. Let the program $\mathring{P}$ satisfy $ground(P) \mapsto_{LX}^* \mathring{P}$. Then $\mathring{P}$ is called a* layered remainder *of P. Since $\mathring{P}$ is equivalent to $\widehat{P}$, apart from the difference between $\mapsto_{LN}$ and $\mapsto_N$, it is trivial that $\mathring{P}$ is also guaranteed to exist and to be unique for P. Moreover, the calculus of $\mapsto_{LX}^*$ is likewise of polynomial time complexity because $\mapsto_{LN}$ is also of polynomial time complexity.*

Example 2. $\mathring{P}$ **versus** $\widehat{P}$**.** Recall the program from example 1 but now with an additional fourth stubborn friend who insists on going to the beach no matter what. $P =$

$$\begin{aligned} beach &\leftarrow not\ mountain \\ mountain &\leftarrow not\ travel \\ travel &\leftarrow not\ beach \\ beach & \end{aligned}$$

We can clearly see that the single fact rule does not depend on any other, and that the remaining three rules forming the loop all depend on each other and on the fact rule $beach$. $\widehat{P}$ is the fixed point of $\mapsto_X$, i.e., the fixed point of $\mapsto_P \cup \mapsto_N \cup \mapsto_S \cup \mapsto_F \cup \mapsto_L$. Since $beach$ is a fact, the $\mapsto_N$ (Negative reduction — definition 9) transformation deletes the $travel \leftarrow not\ beach$ rule; i.e., $P \mapsto_N P'$ is such that

$P' = \{beach \leftarrow not\ mountain \quad mountain \leftarrow not\ travel \quad beach \leftarrow\}$

Now in P' there are no rules for $travel$ and hence we can apply the $\mapsto_P$ (Positive reduction — definition 8) transformation which deletes the $not\ travel$ from the body of $mountain$'s rule; i.e, $P' \mapsto_P P''$ where

$P'' = \{beach \leftarrow not\ mountain \quad mountain \leftarrow \quad beach \leftarrow\}$

Finally, in P'' $mountain$ is a fact and hence we can again apply the $\mapsto_N$ obtaining $P'' \mapsto_P P'''$ where $P''' = \{mountain \leftarrow \quad beach \leftarrow\}$ upon which no more transformations can be applied, so $\widehat{P} = P'''$. Instead, $\mathring{P} = P$ is the fixed point of $\mapsto_{LX}$, i.e., the fixed point of $\mapsto_P \cup \mapsto_{LN} \cup \mapsto_S \cup \mapsto_F \cup \mapsto_L$.

4 Minimal Hypotheses Semantics

4.1 Choosing Hypotheses

The abductive perspective of [10] depicts the atoms of default negated literals (DNLs) as abducibles, i.e., assumable hypotheses. Atoms of DNLs can be considered as abducibles, i.e., assumable hypotheses, but not all of them. When we have a locally stratified program we cannot really say there is any degree of freedom in assuming truth values for the atoms of the program's DNLs. So, we realize that only the atoms of DNLs involved in non-well-founded negation[2] are eligible to be considered further assumable hypotheses.

Both the Stable Models and the approach of [10], when taking the abductive perspective, adopt negative hypotheses only. This approach works fine for some instances of non-well-founded negation such as loops (in particular, for even loops over negation like this one), but not for odd loops over negation like, e.g. $a \leftarrow not\ a$: assuming $not\ a$ would lead to the conclusion that a is *true* which contradicts the initial assumption. To overcome this problem, we generalized the hypotheses assumption perspective to allow the adoption, not only of negative hypotheses, but also of positive ones. Having taken this generalization step we realized that positive hypotheses assumption alone is sufficient to address all situations, i.e., there is no need for both positive and negative hypotheses assumption. Indeed, because we minimize the positive hypotheses we are with one stroke maximizing the negative ones, which has been the traditional way of dealing with the CWA, and also with stable models because the latter's requirement of classical support minimizes models.

In example 1 we saw three solutions, each assuming as *true* one of the DNLs in the loop. Adding a fourth stubborn friend insisting on going to the beach, as in example 2, should still permit the two solutions $\{beach, mountain, not\ travel\}$ and

[2] By non-well-founded negation we mean Strongly Connected Components of rules with at least one head of a rule appearing as a DNL in some body of a rule of the SCC (cf., e.g., Examples 1 and 2).

$\{travel, not\ mountain, beach\}$. The only way to permit both these solutions is by resorting to the Layered Remainder, and not to the Remainder, as a means to identify the set of assumable hypotheses.

Thus, all the literals of P that are not determined *false* in $\mathring{P}$ are candidates for the role of hypotheses we may consider to assume as *true*. Merging this perspective with the abductive perspective of [10] (where the DNLs are the abducibles) we come to the following definition of the Hypotheses set of a program.

Definition 18. ***Hypotheses set of a program.*** *Let P be an NLP. We write $Hyps(P)$ to denote the set of assumable hypotheses of P: the atoms that appear as default negated literals in the bodies of rules of $\mathring{P}$. Formally, $Hyps(P) = \{a : \exists_{r \in \mathring{P}} not\ a \in body(r)\}$.*

One can define a classical support compatible version of the Hypotheses set of a program, only using to that effect the Remainder instead of the Layered Remainder. I.e.,

Definition 19. ***Classical Hypotheses set of a program.*** *Let P be an NLP. We write $CHyps(P)$ to denote the set of assumable hypotheses of P consistent with the classical notion of support: the atoms that appear as default negated literals in the bodies of rules of $\widehat{P}$. Formally, $CHyps(P) = \{a : \exists_{r \in \widehat{P}} not\ a \in body(r)\}$.*

Here we take the layered support compatible approach and, therefore, we will use the Hypotheses set as in definition 18. Since $CHyps(P) \subseteq Hyps(P)$ for every NLP P, there is no generality loss in using $Hyps(P)$ instead of $CHyps(P)$, while using $Hyps(P)$ allows for some useful semantics properties examined in the sequel.

4.2 Definition

Intuitively, a Minimal Hypotheses model of a program is obtained from a minimal set of hypotheses which is sufficiently large to determine the truth-value of all literals via Remainder.

Definition 20. ***Minimal Hypotheses model.*** *Let P be an NLP. Let $Hyps(P)$ be the set of assumable hypotheses of P (cf. definition 18), and H some subset of $Hyps(P)$.*

A 2-valued model M of P is a Minimal Hypotheses model of P iff

$$M^+ = facts(\widehat{P \cup H}) = heads(\widehat{P \cup H})$$

where $H = \emptyset$ or H is non-empty set-inclusion minimal (the set-inclusion minimality is considered only for non-empty Hs). I.e., the hypotheses set H is minimal but sufficient to determine (via Remainder) the truth-value of all literals in the program.

We already know that $WFM^+(P) = facts(\widehat{P})$ and that $WFM^{+u}(P) = heads(\widehat{P})$. Thus, whenever $facts(\widehat{P}) = heads(\widehat{P})$ we have $WFM^+(P) = WFM^{+u}(P)$ which means $WFM^u(P) = \emptyset$. Moreover, whenever $WFM^u(P) = \emptyset$ we know, by Corollary 5.6 of [6], that the 2-valued model M such that $M^+ = facts(\widehat{P})$ is the unique stable model of P. Thus, we conclude that, as an alternative equivalent definition, M is a Minimal Hypotheses model of P iff M is a stable model of $P \cup H$ where H is empty or a non-empty set-inclusion minimal subset of $Hyps(P)$. Moreover, it follows immediately that every SM of P is a Minimal Hypotheses model of P.

In example 2 we can thus see that we have the two models $\{beach, mountain, not\ travel\}$ and $\{travel, beach, not\ mountain\}$. This is the case because the addition of the fourth stubborn friend does not change the set of $Hyps(P)$ which is based upon the Layered Remainder operator, and not on the Remainder one.

4.3 Properties

The minimality of H is not sufficient to ensure minimality of $M^+ = facts(\widehat{P \cup H})$ making its checking explicitly necessary if that is so desired. Minimality of hypotheses is indeed the common practice is science, not the minimality of their inevitable consequences. To the contrary, the more of these the better because it signifies a greater predictive power.

In Logic Programming model minimality is a consequence of definitions: the T operator in definite programs is conducive to defining a least fixed point, a unique minimal model semantics; in SM, though there may be more than one model, minimality turns out to be a property because of the stability (and its attendant classical support) requirement; in the WFS, again the existence of a least fixed point operator affords a minimal (information) model. In abduction too, minimality of consequences is not a caveat, but rather minimality of hypotheses is, if that even. Hence our approach to LP semantics via MHS is novel indeed, and insisting instead on positive hypotheses establishes an improved and more general link to abduction and argumentation [14, 15].

Theorem 1. ***At least one Minimal Hypotheses model of*** P ***complies with the Well-Founded Model.*** *Let P be an NLP. Then, there is at least one Minimal Hypotheses model M of P such that $M^+ \supseteq WFM^+(P)$ and $M^+ \subseteq WFM^{+u}(P)$.*

Proof. If $facts(\widehat{P}) = heads(\widehat{P})$ or equivalently, $WFM^u(P) = \emptyset$, then M_H is a MH model of P given that $H = \emptyset$ because $M_H^+ = facts(\widehat{P \cup H}) = heads(\widehat{P \cup H}) = facts(\widehat{P \cup \emptyset}) = heads(\widehat{P \cup \emptyset}) = facts(\widehat{P}) = heads(\widehat{P})$. On the other hand, if $facts(\widehat{P}) \neq heads(\widehat{P})$, then there is at least one non-empty set-inclusion minimal set of hypotheses $H \subseteq Hyps(P)$ such that $H \supseteq facts(P)$. The corresponding M_H is, by definition, a MH model of P which is guaranteed to comply with $M_H^+ \supseteq WFM^+(P) = facts(\widehat{P})$ and $M_H^- \supseteq not\ WFM^-(P) = not\ (\mathcal{H}_P \setminus M_H^+)$.

Theorem 2. ***Minimal Hypotheses semantics guarantees model existence.*** *Let P be an NLP. There is always, at least, one Minimal Hypotheses model of P.*

Proof. It is trivial to see that one can always find a set $H \subseteq Hyps(P)$ such that $M_{H'}^+ = facts(\widehat{P \cup H'}) = heads(\widehat{P \cup H'})$ — in the extreme case, $H' = Hyps(P)$. From such H' one can always select a minimal subset $H \subset H$ such that $M_H^+ = facts(\widehat{P \cup H}) = heads(\widehat{P \cup H})$ still holds.

4.4 Relevance

Theorem 3. ***Minimal Hypotheses semantics enjoys Relevance.*** *Let P be an NLP. Then, by definition 5, it holds that*

$$(\forall_{M \in Models_{MH}(P)} a \in M^+) \Leftrightarrow (\forall_{M_a \in Models_{MH}(Rel_P(a))} a \in M_a^+)$$

Proof. $\Rightarrow$: Assume $\forall_{M \in Models_{MH}(P)} a \in M^+$. Now we need to prove $\forall_{M_a \in Models_{MH}(Rel_P(a))} a \in M_a^+$. Assume some $M_a \in Models_{MH}(Rel_P(a))$; now we show that assuming $a \notin M_a^+$ leads to an absurdity. Since M_a is a 2-valued complete model of $Rel_P(a)$ we know that $|M_a| = \mathcal{H}_{Rel_P(a)}$ hence, if $a \notin M_a$, then necessarily $not\ a \in M_a^-$. Since $P \supseteq Rel_P(a)$, by theorem 2 we know that there is some model M' of P such that $M' \supseteq M_a$, and thus $not\ a \in M'^-$ which contradicts the initial assumption that $\forall_{M \in Models_{MH}(P)} a \in M^+$. We conclude $a \notin M_a$ cannot hold, i.e., $a \in M_a$ must hold. Since $a \in M^+$ hold for every model M of P, then $a \in M_a$ must hold for every model M_a of $Rel_P(a)$.

$\Leftarrow$: Assume $\forall_{M_a \in Models_{MH}(Rel_P(a))} a \in M_a^+$. Now we need to prove $\forall_{M \in Models_{MH}(P)} a \in M^+$. Let us write $P_{)a(}$ as an abbreviation of $P \setminus Rel_P(a)$. We have therefore $P = P_{)a(} \cup Rel_P(a)$. Let us now take $P_{)a(} \cup M_a$. We know that every NLP as an MH model, hence every MH model M of $P_{)a(} \cup M_a$ is such that $M \supseteq M_a$. Let H_{M_a} denote the Hypotheses set of M_a — i.e., $M_a^+ = facts(\widehat{Rel_P(a) \cup H_{M_a}}) = heads(\widehat{Rel_P(a) \cup H_{M_a}})$, with $H_{M_a} = \emptyset$ or non-empty set-inclusion minimal, as per definition 20. If $facts(\widehat{P \cup H_{M_a}}) = heads(\widehat{P \cup H_{M_a}})$ then $M^+ = facts(\widehat{P \cup H_M}) = heads(\widehat{P \cup H_M})$ is an MH model of P with $H_M = H_{M_a}$ and, necessarily, $M \supseteq M_a$.

If $facts(\widehat{P \cup H_{M_a}}) \neq heads(\widehat{P \cup H_{M_a}})$ then, knowing that every program has a MH model, we can always find an MH model M of $P_{)a(} \cup M_a$, with $H' \subseteq Hyps(P_{)a(} \cup M_a)$, where $M^+ = facts(\widehat{P \cup H'}) = heads(\widehat{P \cup H'})$. Such M is thus $M^+ = facts(\widehat{P \cup H_M}) = heads(\widehat{P \cup H_M})$ where $H_M = H_{M_a} \cup H'$, which means M is a MH model of P with $M \supseteq M_a$. Since every model M_a of $Rel_P(a)$ is such that $a \in M_a^+$, then every model M of P must also be such that $a \in M$.

4.5 Cumulativity

MH semantics enjoys Cumulativity thus allowing for lemma storing techniques to be used during computation of answers to queries.

Theorem 4. ***Minimal Hypotheses semantics enjoys Cumulativity.*** *Let P be an NLP. Then*

$$\forall_{a,b \in \mathcal{H}_\mathcal{P}} \big((\forall_{M \in Models_{MH}(\mathcal{P})} a \in M^+) \Rightarrow (\forall_{M \in Models_{MH}(\mathcal{P})} b \in M^+ \Leftrightarrow \forall_{M_a \in Models_{MH}(\mathcal{P} \cup \{a\})} b \in M_a^+) \big)$$

Proof. Assume $\forall_{\substack{a \in \mathcal{H}_\mathcal{P} \\ M \in Models_{MH}(\mathcal{P})}} a \in M^+$.

$\Rightarrow$: Assume $\forall_{M \in Models_{MH}(\mathcal{P})} b \in M^+$. Since every MH model M contains a it follows that all such M are also MH models of $P \cup \{a\}$. Since we assumed $b \in M$ as well, and we know that M is a MH model of $P \cup \{a\}$ we conclude b is also in those MH models M of $P \cup \{a\}$. By adding a as a fact we have necessarily $Hyps(P \cup \{a\}) \subseteq Hyps(P)$ which means that there cannot be more MH models for $P \cup \{a\}$ than for P. Since we already know that for every MH model M of P, M is also a MH model of $P \cup \{a\}$ we must conclude that $\forall_{M \in Models_{MH}(P)} \exists^1_{M' \in Models_{MH}(P \cup \{a\})}$ such that $M'^+ \supseteq M^+$. Since $\forall_{M \in Models_{MH}(\mathcal{P})} b \in M^+$ we necessarily conclude $\forall_{M_a \in Models_{MH}(\mathcal{P} \cup \{a\})} b \in M_a^+$.

$\Leftarrow$: Assume $\forall_{M_a \in Models_{MH}(\mathcal{P} \cup \{a\})} b \in M_a^+$. Since the MH semantics is relevant (theorem 3) if b does not depend on a then adding a as a fact to P or not has no impact on b's truth-value, and if $b \in M_a^+$ then $b \in M^+$ as well. If b does depend on a, which is true in every MH model M of P, then either 1) b depends positively on a, and in this case since $a \in M$ then $b \in M$ as well; or 2) b depends negatively on a, and in this case the lack of a as a fact in P can only contribute, if at all, to make b true in M as well. Then we conclude $\forall_{M \in Models_{MH}(\mathcal{P})} b \in M^+$.

4.6 Complexity

The complexity issues usually relate to a particular set of tasks, namely: 1) knowing if the program has a model; 2) if it has any model entailing some set of ground literals (a query); 3) if all models entail a set of literals. In the case of MH semantics, the answer to the first question is an immediate "yes" because MH semantics guarantees model existence for NLPs; the second and third questions correspond (respectively) to Brave and Cautious Reasoning, which we now analyse.

Brave Reasoning The complexity of the Brave Reasoning task with MH semantics, i.e., finding an MH model satisfying some particular set of literals is Σ_2^P-complete.

Theorem 5. ***Brave Reasoning with MH semantics is Σ_2^P-complete.*** *Let P be an NLP, and Q a set of literals, or* query. *Finding an MH model such that $M \supseteq Q$ is a Σ_2^P-complete task.*

Proof. To show that finding a MH model $M \supseteq Q$ is Σ_2^P-complete, note that a nondeterministic Turing machine with access to an NP-complete oracle can solve the problem as follows: nondeterministically guess a set H of hypotheses (i.e., a subset of $Hyps(P)$). It remains to check if H is empty or non-empty minimal such that $M^+ = facts(\widehat{P \cup H}) = heads(\widehat{P \cup H})$ and $M \supseteq Q$. Checking that $M^+ = facts(\widehat{P \cup H}) = heads(\widehat{P \cup H})$ can be done in polynomial time (because computing $\widehat{P \cup H}$ can be done in polynomial time [2] for whichever $P \cup H$), and checking H is empty or non-empty minimal requires a nondeterministic guess of a strict subset H' of H and then a polynomial check if $facts(\widehat{P \cup H'}) = heads(\widehat{P \cup H'})$.

Cautious Reasoning Conversely, the Cautious Reasoning, i.e., guaranteeing that every MH model satisfies some particular set of literals, is Π_2^P-complete.

Theorem 6. ***Cautious Reasoning with MH semantics is Π_2^P-complete.*** *Let P be an NLP, and Q a set of literals, or* query. *Guaranteeing that all MH models are such that $M \supseteq Q$ is a Π_2^P-complete task.*

Proof. Cautious Reasoning is the complement of Brave Reasoning, and since the latter is Σ_2^P-complete (theorem 5), the former must necessarily be Π_2^P-complete.

The set of hypotheses $Hyps(P)$ is obtained from $\mathring{P}$ which identifies rules that depend on themselves. The hypotheses are the atoms of default negated literals of $\mathring{P}$, i.e., the "atoms of *not*s in loop". A Minimal Hypotheses model is then obtained from a minimal set of these hypotheses sufficient to determine the 2-valued truth-value of every literal in the program. The MH semantics imposes no ordering or preference between hypotheses — only their set-inclusion minimality. For this reason, we can think of the choosing of a set of hypotheses yielding a MH model as finding a minimal solution to a disjunction problem, where the disjuncts are the hypotheses. In this sense, it is therefore understandable that the complexity of the reasoning tasks with MH semantics is in line with that of, e.g., reasoning tasks with SM semantics with Disjunctive Logic Programs, i.e, Σ_2^P-complete and Π_2^P-complete.

4.7 Comparisons

As we have seen all stable models are MH models. Since MH models are always guaranteed to exist for every NLP (cf. theorem 2) and SMs are not, it follows immediately that the Minimal Hypotheses semantics is a strict model conservative generalization of the Stable Models semantics. The MH models that are stable models are exactly those in which all rules are classically supported. With this criterion one can conclude whether some program does not have any stable models. For Normal Logic Programs, the Stable Models semantics coincides with the Answer-Set semantics (which is a generalization of SMs to Extended Logic Programs), where the latter is known (cf. [8]) to correspond to Reiter's default logic. Hence, all Reiter's default extensions have a corresponding Minimal Hypotheses model. Also, since Moore's expansions of an autoepistemic theory [11] are known to have a one-to-one correspondence with the stable models of the NLP version of the theory, we conclude that for every such expansion there is a matching Minimal Hypotheses model for the same NLP.

As shown in theorem 1, at least one MH model of a program complies with its well-founded model, although not necessarily all MH models do. E.g., the program in Ex. 2 has the two MH models $\{beach, mountain, not\ travel\}$ and $\{beach,$ $not\ mountain, travel\}$, whereas the $WFM(P)$ imposes $WFM^+(P) = \{beach,$ $mountain\}$, $WFM^u(P) = \emptyset$, and $WFM^-(P) = \{travel\}$. This is due to the set of Hypotheses $Hyps(P)$ of P being taken from $\mathring{P}$ (based on the layered support notion) instead of being taken from $\widehat{P}$ (based on the classical notion of support).

Not all Minimal Hypotheses models are Minimal Models of a program. The rationale behind MH semantics is minimality of hypotheses, but not necessarily minimality of consequences, the latter being enforceable, if so desired, as an additional requirement, although at the expense of increased complexity.

The relation between logic programs and argumentation systems has been considered for a long time now ([5] amongst many others) and we have also taken steps to understand and further that relationship [14–16]. Dung's Preferred Extensions [5] are maximal sets of negative hypotheses yielding consistent models. Preferred Extensions, however, these are not guaranteed to always yield 2-valued complete models. Our previous approaches [14, 15] to argumentation have already addressed the issue of 2-valued model existence guarantee, and the MH semantics also solves that problem by virtue of positive, instead of negative, hypotheses assumption.

5 Conclusions and Future Work

Taking a positive hypotheses assumption approach we defined the 2-valued Minimal Hypotheses semantics for NLPs that guarantees model existence, enjoys relevance and cumulativity, and is also a model conservative generalization of the SM semantics. Also, by adopting positive hypotheses, we not only generalized the argumentation based approach of [5], but the resulting MH semantics lends itself naturally to abductive reasoning, it being understood as hypothesizing plausible reasons sufficient for justifying given observations or supporting desired goals. We also defined the layered support notion which generalizes the classical one by recognizing the special role of loops.

For query answering, the MH semantics provides mainly three advantages over the SMs: 1) by enjoying Relevance top-down query-solving is possible, thereby circumventing whole model computation (and grounding) which is unavoidable with SMs; 2) by considering only the relevant sub-part of the program when answering a query it is possible to enact grounding of only those rules, if grounding is really desired, whereas with SM semantics whole program grounding is, once again, inevitable — grounding is known to be a major source of computational time consumption; MH semantics, by enjoying Relevance, permits curbing this task to the minimum sufficient to answer a query; 3) by enjoying Cumulativity, as soon as the truth-value of a literal is determined in a branch for the top query it can be stored in a table and its value used to speed up the computations of other branches within the same top query.

Goal-driven abductive reasoning is elegantly modelled by top-down abductive-query-solving. By taking a hypotheses assumption approach, enjoying Relevance, MH semantics caters well for this convenient problem representation and reasoning category.

Many applications have been developed using the Stable Model/Answer-set semantics as the underlying platform. These generally tend to be focused on solving problems that require complete knowledge, such as search problems where all the knowledge represented is relevant to the solutions. However, as Knowledge Bases increase in size and complexity, and as merging and updating of KBs becomes more and more common, e.g. for Semantic Web applications, [9], partial knowledge problem solving importance grows, as the need to ensure overall consistency of the merged/updated KBs.

The Minimal Hypotheses semantics is intended to, and can be used in *all* the applications where the Stable Models/Answer-Sets semantics are themselves used to model KRR and search problems, *plus* all applications where query answering (both under a credulous mode of reasoning and under a skeptical one) is intented, *plus* all applications where abductive reasoning is needed. The MH semantics aims to be a sound theoretical platform for 2-valued (possibly abductive) reasoning with logic programs.

Much work still remains to be done that can be rooted in this platform contribution. The general topics of using non-normal logic programs (allowing for negation, default and/or explicit, in the heads of rules) for Belief Revision, Updates, Preferences, etc., are *per se* orthogonal to the semantics issue, and therefore, all these subjects can now be addressed with Minimal Hypotheses semantics as the underlying platform. Importantly, MH can guarantee the liveness of updated and self-updating LP programs such as those of EVOLP [1] and related applications. The Minimal Hypotheses semantics still has to be thoroughly compared with Revised Stable Models [13], PStable Models [12], and other related semantics.

In summary, we have provided a fresh platform on which to re-examine ever present issues in Logic Programming and its uses, which purports to provide a natural continuation and improvement of LP development.

References

1. J.J. Alferes, A. Brogi, J. A. Leite, and L. M. Pereira. Evolving logic programs. In S. Flesca et al., editor, *Procs. JELIA'02*, volume 2424 of *LNCS*, pages 50–61. Springer, 2002.
2. S. Brass, J. Dix, B. Freitag, and U. Zukowski. Transformation-based bottom-up computation of the well-founded model. *TPLP*, 1(5):497–538, 2001.
3. J. Dix. A Classification-Theory of Semantics of Normal Logic Programs: II. Weak Properties. *Fundamenta Informaticae*, XXII(3):257–288, 1995.
4. J. Dix. A classification theory of semantics of normal logic programs: I. strong properties, 1995.
5. P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *AI*, 77(2):321–358, 1995.
6. A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
7. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Procs. ICLP'88*, pages 1070–1080, 1988.
8. M. Gelfond and V. Lifschitz. Logic programs with classical negation. In D. Warren et al., editor, *ICLP*, pages 579–597. MIT Press, 1990.
9. A. S. Gomes, J. J. Alferes, and T. Swift. Implementing query answering for hybrid mknf knowledge bases. In M. Carro et al., editor, *Procs. PADL'10*, volume 5937 of *LNCS*, pages 25–39. Springer, 2010.
10. A. C. Kakas, R. A. Kowalski, and F. Toni. Abductive logic programming. *J. Log. Comput.*, 2(6):719–770, 1992.
11. R. C. Moore. Semantical considerations on nonmonotonic logic. *Artif. Intell.*, 25(1):75–94, 1985.
12. M. Osorio and J. C. Nieves. Pstable semantics for possibilistic logic programs. In *Mexican Intl. Conf. on AI, MICAI'07*, volume 4827 of *LNCS*, pages 294–304. Springer, 2007.
13. L. M. Pereira and A. M. Pinto. Revised stable models - a semantics for logic programs. In C. Bento et al., editor, *Procs. EPIA'05*, volume 3808 of *LNAI*, pages 29–42. Springer, 2005.
14. L. M. Pereira and A. M. Pinto. Approved models for normal logic programs. In N. Dershowitz and A. Voronkov, editors, *Procs. LPAR'07*, volume 4790 of *LNAI*. Springer, 2007.
15. L. M. Pereira and A. M. Pinto. Reductio ad absurdum argumentation in normal logic programs. In G. Simari et al., editor, *Procs. Workshop on Argumentation and Non-Monotonic Reasoning (ArgNMR'07), at Conf. Logic Programming and Non-Monotonic Reasoning (LPNMR'07)*, pages 96–113. Springer, 2007.
16. L. M. Pereira and A. M. Pinto. *Oppositional Concepts in Computational Intelligence*, chapter Collaborative vs. Conflicting Learning, Evolution and Argumentation. Studies in Computational Intelligence 155. Springer, 2008.
17. A. M. Pinto. *Every normal logic program has a 2-valued semantics: theory, extensions, applications, implementations*. PhD thesis, Faculdade de Ciências e Tecnologia - Universidade Nova de Lisboa, 2011.
18. R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.