Inductive Tight Semantics for Logic Programs

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

Centro de Inteligência Artificial (CENTRIA)
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal

Abstract. The Gelfond-Lifschitz operator fixed-point requirement by Stable Models induces asymmetry in dealing with Even Loops and Odd Loops Over Negation. We introduce a 2-valued semantics for Normal Logic Programs — the Inductive Tight semantics (ITS) — that generalizes SM semantics by dealing uniformly with Even and Odd loops.ITS conservatively extends the SM semantics (all SMs are Tight models), enjoys relevance and cumulativity, guarantees model existence, and respects the Well-Founded Model.
The IT semantics relies on Layering, a generalization of Stratification, and is inductively defined on such layering: each model for a given layer must comply with some model for the whole set of layers below.
Enjoying Relevance, the IT semantics is suitable top-down querying (*à la* Prolog) when complete models are unnecessary.
The applications afforded by ITS are all those of Stable Models, which it generalizes, plus those employing OLONs for productively obtaining problem solutions, not just filtering them (like Integrity Constraints).
Keywords: Normal Logic Programs, Relevance, Cumulativity, Stable Models, Well-Founded Semantics, Program Remainder

1 Introduction and Motivation

The semantics of Stable Models (SM) [5] is a cornerstone for some of the most important results in logic programming of the past three decades, providing increased logic programming declarativity and a new paradigm for program evaluation. When needing to know the 2-valued truth-value of all literals in a normal logic program (NLP) for the problem being solved, the solution is to produce complete models. In such cases, tools like *SModels* [10] or *DLV* [7] may be adequate enough, as they can indeed compute finite complete models according to the SM semantics and its extensions to Answer Sets [8] and Disjunction. However, lack of some important properties of the base SM semantics, like relevance, cumulativity and guarantee of model existence—enjoyed by, say, Well-Founded Semantics [4] (WFS)—somewhat reduces its applicability and practical ease of use when complete models are unnecessary, and top-down querying (*à la* Prolog) would be sufficient. In addition, abduction by need top-down querying is not an option with SM, creating encumbrance in required pre- and post-processing, because needless full abductive models are generated. The user should not pay the price of computing whole models, nor that of generating all possible abductions and then filtering irrelevant ones, when not needed. Finally, one would like to have available a semantics for that provides a model for every NLP.

WFS in turn does not produce 2-valued models though these are often desired, nor does it guarantee 2-valued model existence.

To overcome these limitations, we present the Inductive Tight Semantics (ITS), a new 2-valued semantics for NLPs which guarantees model existence; preserves the models of SM; enjoys relevance and cumulativity; and complies with the WFM. ITS proffers an alternative to SM-based Answer-Set Programming.

ITS supersedes our previous RSM semantics [9], which we have recently found wanting in capturing our intuitively desired models in some examples, and because ITS relies on a clearer, simpler way of tackling the difficult problem of assigning a semantics to every NLP while affording the aforementioned properties, via adapting better known formal LP methods than RSM's *reductio ad absurdum* stance.

An IT Model (ITM) of an NLP P is any minimal model (MM) M of P that further satisfies $\widehat{P}$—the program remainder of P—in that each loop in $\widehat{P}$ has a MM contained in M, whilst respecting the constraints imposed by the MMs of the other loops so-constrained too.

A couple of examples bring out the need for a semantics supplying all NLPs with models, and permitting models otherwise eliminated by Odd Loops Over default Negation (OLONs):

Example 1. **Jurisprudential reasoning.** A murder suspect not preventively detained is likely to destroy evidence, and in that case the suspect shall be preventively detained:

$$\begin{array}{ll} \mathit{likely\_destroy\_evidence}(\mathit{suspect}) & \leftarrow \mathit{not}\ \mathit{preventive\_detain}(\mathit{suspect}) \\ \mathit{preventive\_detain}(\mathit{suspect}) & \leftarrow \mathit{likely\_destroy\_evidence}(\mathit{suspect}) \end{array}$$

There is no SM, and a single $ITM = \{\mathit{preventive\_detain}(\mathit{suspect})\}$. This jurisprudential reasoning is carried out without need for a *suspect* to exist now. Should we wish, ITS's cumulativity allows adding the model literal as a fact.

Example 2. **A joint vacation problem.** Three friends are planning a joint vacation. First friend says "I want to go to the mountains, but if that's not possible then I'd rather go to the beach". The second friend says "I want to go traveling, but if that's not possible then I'd rather go to the mountains". The third friend says "I want to go to the beach, but if that's not possible then I'd rather go traveling". However, traveling is only possible if the passports are OK. They are OK if they are not expired, and they are expired if they are not OK. We code this information as the NLP:

$$\begin{array}{ll} \mathit{beach} & \leftarrow \mathit{not}\ \mathit{mountain} \\ \mathit{mountain} & \leftarrow \mathit{not}\ \mathit{travel} \\ \mathit{travel} & \leftarrow \mathit{not}\ \mathit{beach}, \mathit{passport\_ok} \\ \mathit{passport\_ok} & \leftarrow \mathit{not}\ \mathit{expired\_passport} \\ \mathit{expired\_passport} & \leftarrow \mathit{not}\ \mathit{passport\_ok} \end{array}$$

The first three rules contain an odd loop over default negation through *beach*, *mountain*, and *travel*; and the rules for *passport_ok* and *expired_passport* form an even loop over default negation. Henceforth we will abbreviate the atoms' names. This program has a single SM: $\{e\_p, m\}$. But looking at the rules relevant for p_ok we find no irrefutable reason to assume e_p to be true. ITS allows p_ok to be true, yielding three other models besides the SM: $ITM_1 = \{b, m, p\_ok\}$, $ITM_2 = \{b, t, p\_ok\}$, and $ITM_3 = \{t, m, p\_ok\}$.

The even loop has two minimal models: $\{p\_ok\}$ and $\{e\_p\}$. Assuming the first MM, the odd loop has three MMs corresponding to ITM_1, ITM_2, and ITM_3 above. Assuming the second MM (where e_p is true), the OLON has only one MM: the SM mentioned above $\{e\_p, m\}$, also an ITM.

The applications afforded by ITS are all those of SM, plus those requiring solving OLONs for model existence, and those where OLONs are employed for the production of solutions, not just used as Integrity Constraints (ICs). Model existence is essential in applications where knowledge sources are diverse (like in the semantic web), and where the bringing together of such knowledge (automatically or not) can give rise to OLONs that would otherwise prevent the resulting program from having a semantics, thereby brusquely terminating the application. A similar situation can be brought about by self-, mutual- and external updating of programs, where unforeseen OLONs would stop short an ongoing process. Coding of ICs via odd loops, commonly found in the literature, can readily be transposed to IC coding in ITS, as explained in the sequel.

Paper structure. After background notation and definitions, we usher in the desiderata for ITS, and only then formally define ITS, exhibit examples, and prove its properties. Conclusions, and future work close the paper.

2 Background Notation and Definitions

Definition 1. ***Normal Logic Program.*** *A Normal Logic Program (NLP) P is a (possibly infinite) set of logic rules, each of the form*

$H \leftarrow B_1, \ldots, B_n, not\ C_1, \ldots, not\ C_m$

where H, the B_i and the C_j are atoms, and each rule stands for all its ground instances. H is the head of the rule, denoted by $head(r)$, and $body(r)$ denotes set $\{B_1, \ldots, B_n, not\ C_1, \ldots, not\ C_m\}$ of all the literals in the body of r. $heads(P)$ denotes $\{head(r) : r \in P\}$. Throughout, 'not ' signals default negation. Abusing notation, we write not S to denote $\{not\ s : s \in S\}$. If the body of a rule is empty, we say its head is a fact and may write the rule just as H.

When S is an arbitrary, non-empty, set of literals we use the following notation: S^+ to denote the subset of positive literals in S, S^- to denote the subset of negative literals in S, and $|S|$ to denote the set of atoms in S — i.e., $|S| = S^+ \cup not\ S^-$.

Definition 2. ***Rule dependencies.*** *Given an NLP P build a dependency graph $G(P)$ such that the rules of P are the nodes of $G(P)$, and there is an* *arc, labeled "positive", from a node r_2 to a node r_1 if head(r_2) appears in the body of r_1; or labeled "negative" if not head(r_2) appears in the body of r_1.*

We say a rule r_1 directly depends on r_2 (written as $r_1 \leftarrow r_2$) iff there is a direct arc in $G(P)$ from r_2 to r_1. By transitive closure we say r_1 depends on r_2 ($r_1 \twoheadleftarrow r_2$) iff there is a path in $G(P)$ from r_2 to r_1.

Definition 3. ***Rule Layering of a Normal Logic Program P.*** *Let P be an NLP, and $G(P)$ its rule-dependency graph. A rule layering function $Lf/1$ is any function defined over the rules of P, assigning each rule $r \in P$ a positive integer, such that the following holds:*

$$\forall_{r_1,r_2 \in P} \begin{cases} Lf(r_1) = Lf(r_2) \Leftarrow (r_1 \twoheadleftarrow r_2) \wedge \quad (r_2 \twoheadleftarrow r_1) \\ Lf(r_1) > Lf(r_2) \Leftarrow (r_1 \leftarrow r_2) \wedge \neg\,(r_2 \twoheadleftarrow r_1) \end{cases}$$

The two cases above, which are patently mutually exclusive, leave out independent rules, i.e., rules that have no dependencies amongst themselves. According to this definition there is no restriction on which integer to assign to each independent rule in what the other rules' assignations are concerned.

A rule layering of program P is a partition $P^0, P^1, \ldots, P^\omega$ of P such that P^i contains all rules r having $Lf(r) = i$, where $P^0 = \emptyset$. Amongst the several possible rule layerings of a program P we can always find the least one, i.e., the rule layering with least number of layers and where the integers of the layers are smallest, whilst respecting the rule layering constraints, easily seen to be unique. In the remainder, when referring to the program's "layering", we mean just such least rule layering. Whenever we want to refer to an atom layering we will explicitly mention "atom" as in "atom layering". We also write $P^{\leq i}$ as an abbreviation of $\bigcup_{0 \leq j \leq i} P^j$. It follows immediately that $P = P^{\leq \omega}$.

Definition 4. ***Loop in P.*** *Given an NLP P, a non-empty subset L of rules of P is a* loop *iff it is maximal (w.r.t. set-inclusion) such that for every two rules $r, r' \in L$, $r \twoheadleftarrow r'$ holds, and there are no two rules $r \neq r' \in L$ such that head$(r) =$ head(r'). We write $Loop_P(L)$ to denote that L is a* loop *of P.*

Rules in a loop are, by definition, placed in the same layer.

Definition 5. ***Set of default negated literals of a loop.*** *Let P be an NLP and L a loop of P. We write nots(L) to denote the set of default negated literals not a in the bodies of rules of L whose positive counterpart a is the head of some rule of L. Formally, nots$(L) = ($not heads$(L)) \cap (\bigcup_{r \in L}$ body$(r)^-)$.*

Definition 6. $Rel_P(a)$ — ***Relevant part of NLP P for the atom a.*** *The Relevant part of a NLP P for some atom a, $Rel_P(a)$ is defined as the transitive closure of the rule-dependency relation of the rules with head a.*

Intuitively, $Rel_P(a)$ is just the set of rules with head a and the rules in the call-graph for a.

Definition 7. ***Program Remainder [2].*** *The program remainder $\widehat{P}$ is guaranteed to exist for every NLP, and is computed by applying to P the* positive reduction *(which deletes the not b from the bodies of rules where b has no rules), the* negative reduction *(which deletes rules that depend on not a where a is a fact), the* success *(which deletes facts from the bodies of rules), and the* failure *(which deletes rules that depend on atoms without rules) transformations, and then eliminating also the* unfounded sets *[4] via a* loop detection *transformation. The* loop detection *is computationally equivalent to finding the strongly connected components [11] in the $G(P)$ graph, as per definition 2, and is known to be of polynomial time complexity.*

Intuitively, unfounded sets consist of positive loops, i.e., loops where the all literals in the bodies of the rules through which the loop is formed are positive.

3 Desiderata

Intuitively desired semantics. Usually, both the default negation *not* and the $\leftarrow$ in rules of Logic Programs reflect some asymmetry in the intended MMs, e.g., in a program with just the rule $a \leftarrow not\ b$, although it has two MMs: $\{a\}$, and $\{b\}$, the only intended one is $\{a\}$. This is afforded by the syntactic asymmetry of the rule, reflected in the one-way direction of the $\leftarrow$, coupled with the intended semantics of default negation. Thus, a fair principle underlying the rationale of a reasonable semantics would be to accept an atom in a model only if there exist rules in a program, at least one, with it as head. This principle rejects $\{b\}$ as a model of program $a \leftarrow not\ b$.

When rules form loops, the syntactic asymmetry disappears and, as far as the loop only is concerned, MMs can reflect the intended semantics of the loop. That is the case, e.g., when we have just the rules $a \leftarrow not\ b$ and $b \leftarrow not\ a$; both $\{a\}$ and $\{b\}$ are the intended models. However, loops may also depend on other literals with which they form no loop. Those asymmetric dependencies should have the same semantics as the single $a \leftarrow not\ b$ rule case described previously.

So, on the one side, asymmetric dependencies should have the semantics of a single $a \leftarrow not\ b$ rule; and the symmetric dependencies (of *any* loop) should subscribe to the same MMs semantics as the $a \leftarrow not\ b$ and $b \leftarrow not\ a$ set of rules. Intuitively, a good semantics should cater for both the symmetric and asymmetric dependencies as described.

Desirable formal properties. By design, our ITS benefits from number of desirable properties of LP semantics [3], namely: guarantee of model existence; relevance; and cumulativity. We recapitulate them here for self-containment. Guarantee of model existence ensures all programs have a semantics. Relevance permits simple (object-level) top-down querying about truth of a query in some model (like in Prolog) without requiring production of a whole model, just the part of it supporting the call-graph rooted on the query. Formally:

Definition 8. ***Relevance.*** *A semantics Sem for logic programs is said Relevant iff for every program P, $a \in Sem(P) \Leftrightarrow a \in Sem(Rel_P(a))$.*

Relevance ensures any partial model supporting the query's truth can always be extended to a complete model; relevance is of the essence to abduction by need, in that only abducibles in the call-graph need be considered for abduction.

Cumulativity signifies atoms true in the semantics can be added as facts without thereby changing it; thus, lemmas can be stored. Formally:

Definition 9. ***Cumulativity.*** *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:*
$Cumulative(Sem) \Leftrightarrow \forall_P \forall_{a,b} a \in Sem(P) \Rightarrow (b \in Sem(P) \Leftrightarrow b \in Sem(P \cup \{a\}))$

Neither of these three properties are enjoyed by SMs, the *de facto* standard semantics for NLPs. The core reason SM semantics fails to guarantee model existence for every NLP is that the stability condition it imposes on models is impossible to be complied with by OLONs.

Example 3. **Stable Models semantics misses Relevance and Cumulativity.**

$$\begin{array}{ll} c \leftarrow not\ c & c \leftarrow not\ a \\ a \leftarrow not\ b & b \leftarrow not\ a \end{array}$$

This program's unique SM is $\{b, c\}$. However, $P \cup \{c\}$ has two SMs $\{a, c\}$, and $\{b, c\}$ rendering b no longer *true* in the SM semantics, which is the intersection of its models. SM semantics lacks Cumulativity. Also, though b is *true* in P according to SM semantics, b is not *true* in $Rel_P(b) = \{a \leftarrow not\ b;\ \ b \leftarrow not\ a\}$, shows SM semantics lacks Relevance.

In fact, the ASP community uses the SM semantics inability to assign a model to OLONs as a means to impose ICs, such as $a \leftarrow not\ a, X$, where the OLON over a prevents X from being *true* in any model.

ITS goes beyond the SM standard, not just because in complying with all the above 3 properties, but also in being a model conservative extension of the SMs semantics, in this sense: A semantics is a model conservative extension of another when it provides at least the same models as the latter, for programs where the latter's are defined, and further provides semantics to programs for which the latter's are not defined. Another way of couching this is: new desired models are provided which the semantics being extended was failing to produce, but all the latter's produced ones are nevertheless provided by the model-conservative extension.

While encompassing the above properties, ITS still respects the Well-Founded Model (WFM) like SM does: every ITS model complies with the true and the false atoms in the WFM of a program. Formally:

Definition 10. ***Well-Founded Model of a Normal Logic Program*** P**.** *Following [2], the* true *atoms of the WFM of P (the irrefutably* true *atoms of P) are the facts of $\widehat{P}$, the* remainder *of P (their definition 5.17). Moreover, the* true *or* undefined *literals of P are just the heads of rules of $\widehat{P}$; and the computation of $\widehat{P}$ can be done in polynomial time. Thus, we shall write $WFM^+(P)$ to denote* *the set of facts of $\widehat{P}$, and $WFM^{+u}(P)$ to denote the set of heads of rules of $\widehat{P}$. Also, since the* false *atoms in the WFM of P are just the atoms of P with no rules in $\widehat{P}$, we write $WFM^{-}(P)$ to denote those false atoms.*

Definition 11. ***Interpretation M of P respects the WFM of P.*** *An interpretation M respects the WFM of P iff M contains the set of all the* true *atoms of the WFM of P, and it contains no* false *atoms of the WFM of P. Formally: $RespectWFM_P(M) \Leftrightarrow WFM^{+}(P) \subseteq M \subseteq WFM^{+u}(P)$*

ITS's WFM compliance, besides keeping with SM's compliance (i.e. the WFM approximates the SM), is important to ITS for a specific implementation reason too. Since WFS enjoys relevance and polynomial complexity, one can use it to obtain top-down—in present day tabled implementations—the residual or remainder program that expresses the WFM, and then apply ITS to garner its 2-valued models, foregoing the need to generate complete models.

For program $a \leftarrow not\ a$, the only Inductive Tight Model (ITM) is $\{a\}$. In the ITS, OLONs are not ICs. ICs are enforced employing rules for the special atom $falsum$, of the form $falsum \leftarrow X$, where X is the body of the IC one wishes to prevent being true. This does not preclude $falsum$ from figuring in some models. From a theoretical standpoint it means the ITS semantics does not *a priori* include a built-in IC compliance mechanism. ICs can be dealt with in two ways, either by (1) a syntactic post-processing step, as a model "test stage" after their "generate stage"; or by (2) embedding IC compliance in the query-driven computation, whereby the user conjoins query goals with $not\ falsum$. If inconsistency examination is desired, like in case (1), models including $falsum$ can be discarded *a posteriori*. Thus, ITS clearly separates OLON semantics from IC compliance, and frees OLONs for a wider knowledge representation usage.

4 Inductive Tight Semantics

The fundamental principle which guided the crafting of the Tightness concept is the *default* character of negative literals in Normal Logic Programs. In a non-definite NLP, whenever there are loops through default negation, negative literals as considered free choices or assumptions. In fact, even from an abductive perspective, [6] depicts how default negated literals (DNLs) can be viewed as abducibles, i.e., assumable hypotheses.

Merging this default character of the *not* with the syntactic symmetries of loops and asymmetries of non mutually dependent rules is what Tightening achieves. For that, Tightness is a two-fold concept which encompasses Symmetric Tightness and Asymmetric Tightness.

4.1 Symmetric Tightness

Symmetric Tightness (ST) implements the semantic symmetry principle which stems from the syntactic symmetry of loops, therefore, ST applies only to rules forming a loop.

Taking the stance of facing each default negated literal (DNL) as an assumable hypothesis, or free choice, the ST reflects the syntactic symmetry of a loop by assigning equal ground to every DNL — no DNL is preferable over any other in the loop in what assuming its truth-value as *true* is concerned. However, once one particular literal a — where $not\ a \in nots(L)$ for the loop L at hand — is assumed *true*, the consequences of that choice must be propagated through L. The Tightness (both the Symmetric and the Asymmetric as we shall see in subsection 4.2) resorts to the Program Remainder operator ($\widehat{P}$, as per def. 7) as a means to compute and propagate such consequences. This restricts which DNLs are still assumable as *true* (and which must necessarily be assumed *false*) after a given DNL has been assumed *true*.

Definition 12. ***Atom Tightened Loop.*** *Let P be an NLP, and L a loop of P, and a an atom of L such that $not\ a \in nots(L)$. The atom tightening of L with a — denoted as $L \circledcirc a$ — is obtained by adding a as a fact to L and then calculating the Program Remainder, i.e. $L \circledcirc a = \widehat{L \cup \{a\}}$.*

Notice that after choosing one particular a, the corresponding $L \circledcirc a$ can be deterministically calculated by a polynomial-time process (which is the know complexity of the Program Remainder operator).

Definition 13. ***Fully Tightened Loop.*** *Let P be an NLP, L a loop of P, and a an atom of L such that $not\ a \in nots(L_i)$, where $L_0 = L$, and $L_{i+1} = L_i \circledcirc a$. A full tightening of L — denoted as $L\circ$ — is an end point of the atom tightening sequence. I.e., $L\circ$ is some L_j where $L_j = L_{j+1}$.*

Each atom tightening step adds a fact to L and reduces L via the Program Remainder operator. After one atom tightening step with, say, atom a, $nots(L \circledcirc a)$ may still be non-empty, and in that case achieving $L\circ$ will require more atom tightening steps. When, after j atom tightening steps, $nots(L_j) = \emptyset$, the original L is reduced to L_j, necessarily a set of facts. This occurs due to the simplifications (rule deletions and rules bodies' reductions) the Program Remainder operator performs after the addition of a fact. Hence, $L\circ$ is just a set of facts. Since, in principle, any sequence of atoms a of $nots(L)$ can be used for calculating a $L\circ$, there may be, at most $2^{\#nots(L)}$ different $L\circ$.

Given an arbitrary loop L with $\#nots(L) = m$, any fully tightened version $L\circ$ of L can be obtained by performing, at most, m atom tightening operations, since m is necessarily finite. Hence, at most, $L\circ = L_m = L\circledcirc^m$.

Definition 14. ***Loop Tight Model.*** *Let P be an NLP, and L a loop of P, and $L\circ$ some full tightening of L. The set M of facts of $L\circ$ is a Loop Tight (LT) model of L — denoted as $LT_L(M)$. If $nots(L) = \emptyset$ then L has one unique LT model: $M = \emptyset$.*

Example 4. **Loop Tight Model.** Let P be some NLP, with the loops

$$\begin{array}{ll} L_1 = & L_2 = \\ \quad a \leftarrow d & \quad w \leftarrow not\ x \\ \quad d \leftarrow not\ b & \quad x \leftarrow not\ y \\ \quad b \leftarrow not\ c & \quad y \leftarrow not\ z \\ \quad c \leftarrow not\ a & \quad z \leftarrow not\ w \end{array}$$

L_1 has three LT models: $M_{L_{1_1}} = \{a, b\}$, $M_{L_{1_2}} = \{b, c\}$, $M_{L_{1_3}} = \{c, a, d\}$.

For $M_{L_{1_1}}$ we start by assuming a *true*. We add a as a fact to L_1, thus obtaining $L_1 \cup \{a\}$, and compute $L_1 \circledcirc a = \widehat{L_1 \cup \{a\}} = \{a, b\}$.

The computation of the Program Remainder of $L_1 \cup \{a\}$ goes as follows: since a is now a fact, the rule $c \leftarrow not\ a$ is deleted; since this was the only rule for c, this deletion renders c *false*, hence the body of the rule $b \leftarrow not\ c$ can be simplified by deleting the $not\ c$. b now becomes a fact too. By the same token, since b is now a fact, the rule $d \leftarrow not\ b$ is deleted and, this being the unique rule for d, the rule $a \leftarrow d$ is also deleted.

$L_1 \circledcirc a$ is thus the set of facts $\{a, b\}$, so $M_{L_{1_1}} = \{a, b\}$. $M_{L_{1_2}}$ and $M_{L_{1_3}}$ can be computed by the same process.

One could suspect L_2 having four LT models: $M_{L_{2_1}} = \{w, y\}$, $M_{L_{2_2}} = \{z, x\}$, $M_{L_{2_3}} = \{y, w\}$, $M_{L_{2_4}} = \{x, z\}$, but it is clear that $M_{L_{2_1}} = M_{L_{2_3}}$ and that $M_{L_{2_2}} = M_{L_{2_4}}$.

Definition 15. ***Layer Tight model.*** *Let P be an NLP, and M a set of literals such that, for each loop L in P there is a $LT_L(M_L)$, where $M_L \subseteq M$. M is a Layer Tight model of P — denoted as $LyT_P(M)$ — iff M^+ is a minimal model of P and $M^- = nots(P) \setminus (not\ M^+)$.*

Theorem 1. ***Existence of Layer Tight Model.*** *Every NLP P has, at least, one Layer Tight Model.*

Proof. For each loop L in P it always possible to find all of its Loop Tight models. Each LT model is, by definition, a model of its respective loop. Each combination of LT models (one LT model per loop) united with the heads of rules of P outside loops, is a model of P. Each such minimal combination, plus the heads of the non-loop rules, is M^+, necessarily a minimal model of P. The set $nots(P)$ always exists and is unique. It is always possible to compute deterministically M^- from $nots(P)$ and M^+. Hence, there is always at least one Layer Tight model for P.

4.2 Asymmetric Tightness

Asymmetric Tightness (AT) implements the semantic asymmetry principle which stems from the syntactic asymmetry of different layers, therefore, AT applies only between layers. Truth-values for literals determined in a given layer must be respected by rules in layers above. The asymmetric tightening guarantees that respect by deleting rules whose bodies are inconsistent with the model determined for the layers below.

Definition 16. ***Layer-wise inconsistency.*** *Let P be an NLP, and S a set of literals. A rule $r \in P$ is* layer-wise inconsistent *with S iff*
$not\ body(r) \cap (S \setminus (heads(P) \cup not\ heads(P))) \neq \emptyset$

Definition 17. ***Asymmetric Tightening.*** *Let P be an NLP, and S a consistent set of literals of P. The Asymmetric Tightening of P by S — $P : S$ — is $P : S = \widehat{P_S}$, where*
$P_S = \{head(r) \leftarrow body(r) \setminus S : r \in P \wedge r$ *is layer-wise consistent with* $S\}$

The layer-wise consistency proviso allows for rules with bodies inconsistent with S to be in $P : S$ as along as the source of inconsistency is limited to the literals of the body for which there are also rules in P. This proviso thus does not prevent Loop Tight models for the loops in P to corroborate the truth-value for the literals already determined *true* in S.

Definition 18. ***Inductive Tight Model.*** *Let P be an NLP. $M_0 = \emptyset$ is the unique Inductive Tight model for $P^0 = P^{\leq 0} = \emptyset$. Assume there is an Inductive Tight model $M_{<\alpha}$ for $P^{<\alpha}$, with $\alpha > 0$. $M_{\leq\alpha}$ is an Inductive Tight model of $P^{\leq\alpha}$ — denoted as $ITM_{P^{\leq\alpha}}(M_{\leq\alpha})$ — iff $\exists_{LyT_{P^{\alpha}:M_{<\alpha}}(M_\alpha)} M_{\leq\alpha} = M_\alpha \cup M_{<\alpha}$.*

The Inductive Tight Semantics of P — $ITS(P)$ — is the intersection of all its Inductive Tight Models.

The rationale for IT models is actually quite simple. IT models are defined in an inductive manner along the layers of the program. Layer 0 has the unique IT model $\emptyset$. Each subsequent layer is Asymmetrically Tightened with some immediately-lower layer IT model, before a Layer Tight model of the (already Asymmetrically Tightened) layer can be calculated.

Example 5. **Difference between ITS and RSM semantics.** Let P be

$$\begin{array}{l} a \leftarrow not\ b, c \\ b \leftarrow not\ c, not\ a \\ c \leftarrow not\ a, b \end{array}$$

ITS accepts both $M_1 = \{a\}$ and $M_2 = \{b, c\}$ as ITMs, whereas the RSM semantics [9] only accepts M_1. Neither are SMs.

Example 6. **Mixed loops 2.** Let P be

$$\begin{array}{l} a \leftarrow not\ b \\ b \leftarrow not\ c, e \\ c \leftarrow not\ a \\ e \leftarrow not\ e, a \end{array}$$

In this case, ITS, like the RSM semantics, accepts all minimal models: $M_1 = \{a, b, e\}$, $M_2 = \{a, c, e\}$, and $M_3 = \{b, c\}$.

Example 7. **Quasi-Stratified Program.** Let P be

$$\begin{aligned} d &\leftarrow not\ c \\ c &\leftarrow not\ b \\ b &\leftarrow not\ a \\ a &\leftarrow not\ a \end{aligned}$$

The unique ITS is $\{a, c\}$, and there are no SMs. In this case it is quite easy to see how the Tightness works: $M_0 = M_{\leq 0} = M_{<1} = \emptyset$. Asymmetrically tightening P^1 with $M_{<1} = \emptyset$ leaves P^1 unchanged. $M_1 = \{a\}$ is necessarily the unique LyT model of $P^1 = a \leftarrow not\ a$. $M_{<2} = M_{\leq 1} = M_1 \cup M_{<1} = M_1 = \{a\}$. Asymmetrically tightening P^2 with $M_{<2}$ produces $P^2 : \{a\} = \emptyset$. The unique Layer Tight model of $P^2 : \{a\}$ is $M_2 = \emptyset$. So, $M_{<3} = M_{\leq 2} = M_2 \cup M_{<2} = \emptyset \cup \{a\} = \{a\}$. Asymmetrically tightening P^3 with $M_{<3}$ produces $P^3 : \{a\} =$ the fact c. Now, the unique Layer Tight model of $P^3 : \{a\}$ is $M_3 = \{c\}$. So, $M_{<4} = M_{\leq 3} = M_3 \cup M_{<3} = \{a\} \cup \{c\} = \{a, c\}$. Finally, Asymmetrically tightening P^4 with $M_{<4}$ produces $P^4 : \{a, c\} = \emptyset$. The unique Layer Tight model of $P^4 : \{a, c\}$ is $M_4 = \emptyset$, and the unique Inductive Tight model of $P = P^{\leq 4}$ is $M_{\leq 4} = M_4 \cup M_{<4} = \emptyset \cup \{a, c\} = \{a, c\}$.

5 Properties of the Inductive Tight Semantics

Forthwith, we prove some properties of ITS, namely: guarantee of model existence, relevance, cumulativity, model-conservative extension of SMs, and respect for the Well-Founded Model. The definitions involved are to be found in section 3.

Theorem 2. ***Existence.*** *Every well-founded Normal Logic Program has an IT model.*

Proof. Let P be an NLP. It is always possible to calculate P's least layering. The inductive definition itself provides the method to prove that all well-founded NLPs have Inductive Tight Models. P^0 has a unique IT model: the empty set. Assuming $M_{<\alpha}$ is an IT model for all layers $\beta < \alpha$, it is always possible to asymmetrically tighten layer α with $M_{<\alpha}$. Theorem 1 guarantees there is always at least one $LyT_{P^{\alpha}:M_{<\alpha}}(M_\alpha)$. And, of course, it is possible to compose $M_{\leq\alpha} = M_{<\alpha} \cup M_\alpha$.

Theorem 3. ***Relevance of IT Semantics.*** *The IT Semantics is relevant.*

Proof. According to definition 8 a semantics Sem is relevant iff $a \in Sem(P) \Leftrightarrow a \in Sem(Rel_P(a))$ for all atoms a. Since the ITS of a program P — $ITS(P)$ — is the intersection of all its ITMs, relevance becomes $a \in ITS(P) \Leftrightarrow a \in ITS(Rel_P(a))$ for ITS.

By definition, a is *true* in some IT model M iff either a is a fact or there is some rule r such that $head(r) = a$ and $r \in L$ for some $L \in P$, in whichever layer. In either case a is *true* in M because there is some rule with head a. Necessarily, such rule belongs to $Rel_P(a)$.

Theorem 4. ***Cumulativity of IT Semantics.*** *The IT Semantics is cumulative.*

Proof. By definition 18, the semantics of a program P is the intersection of its ITMs. So, $a \in ITS(P) \Leftrightarrow \forall_{ITM_P(M)} a \in M$. For the ITS semantics cumulativity becomes expressed by
$Cumulative(Sem) \Leftrightarrow \forall_{a,b} a \in ITS(P) \Rightarrow (b \in ITS(P) \Leftrightarrow b \in ITS(P \cup \{a\}))$

Let us assume $a \in ITS(P) \wedge b \in ITS(P)$. Adding an atom as a fact to P consists in putting it in the lowest layer. Since the ITS is relevant, if b did not depend on a, adding a to P will not affect b's truth-value. The only way for the addition of a as a fact to P to prevent b from being in $ITS(P)$ would be by a preventing any LT model from containing b. But this cannot be the case precisely because the Asymmetric Tightening does not allow for the deletion of rules with bodies inconsistent with literals determined in lower layers as long as the rule with the conflicting body is in a loop with another rule whose head is one of the conflicting literals.

Let us assume $a \in ITS(P) \wedge b \in ITS(P \cup \{a\})$. Again, if b does not depend on a, since the semantics is relevant, b's truth-value remains unchanged. By the same reason, invoked in the previous paragraph concerning of the Asymmetric Tightening, b will necessarily remain in $ITS(P)$.

Theorem 5. ***Stable Models Extension.*** *Any Stable Model is an ITM of P.*

Proof. Assume M is a SM of P. Then $M = least(P/M)$ where the division P/M deletes all rules with *not* a in the body where $a \in M$, and then deletes all remaining *not* b from the bodies of rules. The program division P/M performs an even more demanding program division than that of Asymmetric Tightening. Moreover, the division P/M acts on all layers at once. Clearly, $M = least(P/M)$ only if it also passes the less demanding test of Inductive Tightness.

Theorem 6. ***Inductive Tight Semantics respects the Well-Founded Model.*** *Every IT Model of P respects the Well-Founded Model of P.*

Proof. Take any ITM M of P. At each layer i, M_i is a minimal model of layer i already asymmetrically tightened by a lower layer set model. The asymmetric tightening performs a Program Remainder computation, which guarantees respect for the WFM. In fact, we resort to the Program Remainder operator in several definitions precisely to guarantee respect for the WFM.

Due to lack of space, the complexity analysis of this semantics is left out of this paper. Nonetheless, a brief note is due. Inductive Tight Model existence is guaranteed for every NLP, whereas finding if there are any SMs for an NLP is NP-complete. Since ITS enjoys relevance, the computational scope of Brave Reasoning can be restricted to $Rel_P(a)$ only, instead of the whole P. Nonetheless, we conjecture that Brave reasoning — finding if there is any model of the program where some atom a is true — is a Σ_P^2-hard task. This is so because each relevant branch in the call-graph can be a loop. Traversing the entire call-graph is in itself an NP-complete task. For each loop, the ITS requires the computation of a minimal model — another NP-complete task. Hence the conjectured Σ_P^2-hardness. Still, from a practical standpoint, having to traverse only the relevant call-graph for brave reasoning, instead of considering the whole program, can have a significant impact in the performance of concrete applications. By the same token, cautious reasoning (finding out if some atom a is in all models) in the ITS should have the complementary complexity of brave reasoning: Π_P^2-complete.

One common objection to this kind of semantics concerns the notion of SM support. IT models are not supported, considering that classical notion of support. However, we abide by a more general notion of support: an atom is loop supported iff there is at least one rule with it as head and all the literals in the body of the rule which do not depend on the head are also true. This loop support is a generalization of the classical support: every SM model also a ITS model is classically and loop supported. This ensures the truth assignment to atoms in, say, a loop L_2 which depends asymmetrically on loop L_1, is consistent with the truth assignments in loop L_1 and that these take precedence over L_2 in their truth labeling. As a consequence of the loop support requirement, IT models comply with the WFM of the loops they asymmetrically depend on.

6 Conclusions, Future and Ongoing Topics, and Similar Work

Having defined a more general 2-valued semantics for LPs much remains in store, and to be explored and reported, in the way of properties, complexity, comparisons, implementation, and applications. We hope the concepts and techniques newly introduced here might be adopted by other logic programming semantics approaches and systems.

We defined ITS, a semantics for *all* NLPs complying with the express requirements of: 2-valued semantics, preserving the models of SM, guarantee of model existence (even in face of odd loops over negation), relevance, cumulativity, and WFM respect.

Relevancy condones top-down querying and avoids the need to compute whole models. It also permits abduction by need, avoiding much useless consideration of irrelevant abducibles.

That ITS includes the SM semantics and that it always exists and admits top-down querying is a novelty making us look anew at 2-valued semantics use in KRR, contrasting its use to other semantics employed heretofore for KRR, even though SM has already been compared often enough [1].

References

1. C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving.* Cambridge University Press, 2003.

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: I, II. *Fundamenta Informaticae*, XXII(3):227–255, 257–288, 1995.
4. A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *J. of ACM*, 38(3):620–650, 1991.
5. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, pages 1070–1080. MIT Press, 1988.
6. Antonis C. Kakas, Robert A. Kowalski, and Francesca Toni. Abductive logic programming. *J. Log. Comput.*, 2(6):719–770, 1992.
7. Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The dlv system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7:499–562, 2002.
8. Vladimir Lifschitz and Thomas Y. C. Woo. Answer sets in general nonmonotonic reasoning (preliminary report). In *KR*, pages 603–614, 1992.
9. L. M. Pereira and A. M. Pinto. Revised stable models - a semantics for logic programs. In G. Dias et al., editor, *Progress in AI*, volume 3808 of *LNCS*, pages 29–42. Springer, 2005.
10. Tommi Syrjänen and Ilkka Niemelä. The smodels system. In T. Eiter et al., editor, *LPNMR 2001*, volume 2173 of *LNAI*. Springer-Verlag, 2001.
11. R. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Computing*, 1(2):146–160, 1972.