Technical Communications of the International Conference on Logic Programming, 2010 (Edinburgh), pp. 134–143
http://www.floc-conference.org/ICLP-home.html

TIGHT SEMANTICS FOR LOGIC PROGRAMS

LUÍS MONIZ PEREIRA [1] AND ALEXANDRE MIGUEL PINTO [1]

[1] Centro de Inteligência Artificial (CENTRIA)
Departamento de Informática, Faculdade de Ciências e Tecnologia,
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
E-mail address: lmp@di.fct.unl.pt

E-mail address: amp@di.fct.unl.pt

Abstract. We define the Tight Semantics (TS), a new semantics for *all* NLPs complying with the requirements of: 2-valued semantics; preserving the models of SM; guarantee of model existence, even in face of Odd Loops Over Negation (OLONs) or infinite chains; relevance; cumulativity; and compliance with the Well-Founded Model.

When complete models are unnecessary, and top-down querying (*à la* Prolog) is desired, TS provides the 2-valued option that guarantees model existence, as a result of its relevance property. Top-down querying with abduction by need is rendered available too by TS. The user need not pay the price of computing whole models, nor that of generating all possible abductions, only to filter irrelevant ones subsequently.

A TS model of a 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.

The applications afforded by TS are all those of Stable Models, which it generalizes, plus those permitting to solve OLONs for model existence, plus those employing OLONs for productively obtaining problem solutions, not just filtering them (like Integrity Constraints).

1. Introduction and Motivation

The semantics of Stable Models (SM) [Gel88] 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* [Syr01] or *DLV* [Leo02] may be adequate enough, as they can indeed compute finite complete models according to the SM semantics and its extensions to Answer Sets [Lif92] 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 [Gel91] (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,

Key words and phrases: Normal Logic Programs, Relevance, Cumulativity, Stable Models, Well-Founded Semantics, Program Remainder.

Technical Communications of the 26th International Conference on Logic Programming, Edinburgh, July, 2010
Editors: Manuel Hermenegildo, Torsten Schaub
LIPIcs - Leibniz International Proceedings in Informatics. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.ICLP.2010.134

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 Tight Semantics (TS), 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. TS also deals with infinite chains [Fag94], proffering an alternative to SM-based Answer-Set Programming.

TS supersedes our previous RSM semantics [Per05], which we have recently found wanting in capturing our intuitively desired models in some examples, and because TS 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.

A TM 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 1.1. Jurisprudential reasoning. A murder suspect not preventively detained is likely to destroy evidence, and in that case the suspect shall be preventively detained:

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

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

Example 1.2. 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:

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

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. TS allows p_ok to be true, yielding three other models besides the SM: $TM_1 = \{b, m, p\_ok\}$, $TM_2 = \{b, t, p\_ok\}$, and $TM_3 = \{t, m, p\_ok\}$.

The even loop has two minimal models: $\{p\_ok\}$ and $\{e\_p\}$. Assuming the first MM, the odd loop has three MMs corresponding to TM_1, TM_2, and TM_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 a TM.

The applications afforded by TS 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 TS, as explained in the sequel.

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

2. Background Notation and Definitions

Definition 2.1. Normal Logic Program. A Normal Logic Program (NLP) P is a (possibly infinite) set of logic rules, each of the form $H \leftarrow B_1, \ldots, B_n, \mathit{not}\, C_1, \ldots, \mathit{not}\, C_m$ 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 $\mathit{head}(r)$, and $\mathit{body}(r)$ denotes set $\{B_1, \ldots, B_n, \mathit{not}\, C_1, \ldots, \mathit{not}\, C_m\}$ of all the literals in the body of r. $\mathit{heads}(P)$ denotes $\{\mathit{head}(r) : r \in P\}$. Throughout, '*not* ' signals default negation. Abusing notation, we write $\mathit{not}\ S$ to denote $\{\mathit{not}\ s : s \in S\}$. If the body of a rule is empty, we say its head is a fact and may write the rule just as H.

Throughout too, we consider MMs of programs, and write $MM_P(M)$ to denote M is a minimal model of P. When both $MM_P(M)$ and $M \subseteq \mathit{heads}(P)$ hold, then M_N denotes the union of M with the negations of heads of P absent in M; i.e., $M_N = M \cup \mathit{not}\ (\mathit{heads}(P) \backslash M)$. We dub M_N a *completed minimal model* of P.

Definition 2.2. 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 $\mathit{head}(r_2)$ appears in the body of r_1; or labeled "negative" if $\mathit{not}\ \mathit{head}(r_2)$ 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.

Dependencies through default negation play a major role in the sequel and so we also need to define the following: we say a rule r_1 directly depends negatively on r_2 (written as $r_1 \leftarrow -r_2$) iff $\mathit{not}\ \mathit{head}(r_2)$ appears in the body of r_1. By transitive closure we say r_1 depends negatively on r_2 ($r_1 \twoheadleftarrow -r_2$) iff r_1 directly depends negatively on r_2 or r_1 depends on some r_3 which directly depends negatively on r_2.

Definition 2.3. $Rel_P(a)$ **— Relevant part of NLP** P **for the positive literal** a. The Relevant part of a NLP P for some positive literal a, $Rel_P(a)$ is defined as

$$Rel_P(a) = \bigcup\{r, r' \in P : r \twoheadleftarrow r' \wedge head(r) = a\}$$

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

Definition 2.4. Loop in P. We say a subset P_L of rules of P is a *loop* iff for every two rules r_1 and r_2 in P_L there is a path from r_1 to r_2 in $G(P)$ and vice-versa. I.e., $\forall_{r_1,r_2 \in P_L} r_1 \twoheadleftarrow r_2 \wedge r_2 \twoheadleftarrow r_1$. We write $Loop(P_L)$ to denote that P_L is a *Loop*.

Definition 2.5. Program Remainder [Bra01]. 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* [Gel91] via a *loop detection* transformation. The *loop detection* is computationally equivalent to finding the strongly connected components [Tar72] in the $G(P)$ graph, as per definition 2.2, and is known to be of polynomial time complexity.

Definition 2.6. Program Division. Let P be an NLP and $I \subseteq heads(P) \cup not\ heads(P)$ a consistent interpretation of P. $P : I$ denotes the subset of P remaining after performing this sequence of steps:

(1) delete rules with *not* a in the body where $a \in I$ – similar to *negative reduction*
(2) delete all a in the bodies of rules where $a \in I$ – similar to *success*
(3) delete all *not* a in the bodies of rules where *not* $a \in I$

The rationale behind program division is to obtain the subset of P remaining after considering all literals in I true. Step 1 eliminates the rules of P which are already satisfied (in a classical way) by the literals in I. Step 2 is similar to *success* but deletes all positive literals a from the bodies of rules where $a \in I$. Step 3 is a negative counterpart of step 2; one could dub it *negative success*. Thus, steps 2 and 3 are slightly more credulous that the original *success*.

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 TS benefits from number of desirable properties of LP semantics [Dix95], 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 3.1. 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 3.2. 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:
$Cumulative(Sem) \Leftrightarrow \forall_P \forall_{a,b\_} a \in Sem(P) \wedge b \in Sem(P) \Rightarrow a \in Sem(P \cup \{b\})$

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 3.3. 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.

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

Definition 3.4. Well-Founded Model of a Normal Logic Program P. Following [Bra01], 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 3.5. 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:

$$\mathit{RespectWFM}_P(M) \Leftrightarrow WFM^{+}(P) \subseteq M \subseteq WFM^{+u}(P)$$

TS's WFM compliance, besides keeping with SM's compliance (i.e. the WFM approximates the SM), is important to TS 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 TS to garner its 2-valued models, foregoing the need to generate complete models.

For program $a \leftarrow \mathit{not}\ a$, the only Tight Model (TM) is $\{a\}$. In the TS, 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 TS 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 $\mathit{not}\ falsum$. If inconsistency examination is desired, like in case (1), models including $falsum$ can be discarded *a posteriori*. Thus, TS clearly separates OLON semantics from IC compliance, and frees OLONs for a wider knowledge representation usage.

4. Tight Semantics

The rationale behind tightness follows the intuitively desired semantics principles described in section 3. On the one side any TM M is necessarily an MM of $\widehat{P}$ which guarantees that no atoms with no rules are in M. This is in accordance with the principle of intuitively desired semantics for asymmetric dependencies, and is also what guarantees that TMs respect the WFM, as proved in the sequel. On the other side, implementing the intuitively desired semantics for symmetric dependencies, the TS imposes each TM to have the internal loop congruency of tightness: the semantics for each loop is its MMs, as long as a chosen MM for it is compatible (via program division) with the model for the whole program, while the rest $M_{\overline{L}}$ of the original model M is itself Tight.

Definition 4.1. Tight Model. Let P be an NLP, and M a minimal model of $\widehat{P}$, such that M_N is a completed minimal model of P. Let $\widehat{P}_L$ denote a $Loop(\widehat{P}_L)$ strictly contained in $\widehat{P}$; and given $MM_{\widehat{P}_L}(M_L)$, let $M_{\overline{L}}$ denote $(M \setminus M_L) \cup \{M_L \cap heads(P : M_{L_N})\}$. We say M is tight in P — $Tight_P(M)$ — iff

$$\exists \widehat{P}_L \Rightarrow \exists M_L : M_{L_N} \subseteq M_N \wedge Tight_{P:M_{L_N}}(M_{\overline{L}})$$

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

Example 4.2. Mixed loops 1. Let P be

$$\begin{array}{lll} a \leftarrow k & k \leftarrow not\ t & t \leftarrow a, b \\ a \leftarrow not\ b & b \leftarrow not\ a & \end{array}$$

$\widehat{P}$ coincides with P, so its MMs are $M_1 = \{a, k\}$ with $M_{1_N} = \{a, not\ b, k, not\ t\}$; $M_2 = \{a, t\}$ with $M_{2_N} = \{a, not\ b, not\ k, t\}$; and $M_3 = \{b, t\}$ with $M_{3_N} = \{not\ a, b, not\ k, t\}$. Of these, only M_{1_N} and M_{3_N} are Tight. M_1 in particular is also an SM. To see that M_{2_N} is not Tight notice that there are three loops in P: $P_{L_1} = \{a \leftarrow not\ b; b \leftarrow not\ a\}$, $P_{L_2} = \{a \leftarrow k; k \leftarrow not\ t; t \leftarrow a, b\}$, $P_{L_3} = \{a \leftarrow k; k \leftarrow not\ t; t \leftarrow a, b; b \leftarrow not\ a\}$. The MMs of P_{L1} are $M_{L1_1} = \{a\}$ with $M_{L1_{1N}} = \{a, not\ b\}$, and $M_{L1_2} = \{b\}$ with $M_{L1_{2N}} = \{not\ a, b\}$. Dividing P by $M_{L1_{1N}}$ we get $P : M_{L1_{1N}} = \{a \leftarrow k; k \leftarrow not\ t; t \leftarrow b\}$. $M_{\overline{L1}_1}$ is now $(\{a, t\} \setminus \{a\}) \cup \{\{a\} \cap heads(\{a \leftarrow k; k \leftarrow not\ t; t \leftarrow b\})\} = \{t\} \cup \{a\} = \{a, t\}$. But $\{a, t\}$ is not Tight in $P : M_{L1_{1N}}$ since it is not even an MM of it.

Example 4.3. Difference between TS and RSM semantics. Let P be

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

TS accepts both $M_1 = \{a\}$ and $M_2 = \{b, c\}$ as TMs, whereas the RSM semantics [Per05] only accepts M_1. Neither are SMs.

Example 4.4. 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, TS, like the RSM semantics, accepts all minimal models: $M_1 = \{a, b, e\}$, $M_2 = \{a, c, e\}$, and $M_3 = \{b, c\}$.

Example 4.5. Quasi-Stratified Program. Let P be

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

The unique TS is $\{a, c\}$, and there are no SMs. In this case it is quite easy to see how the Tightness works: $\{a\}$ is necessarily the unique MM of $a \leftarrow not\ a$. Dividing the whole program by $\{a\}$ we get $\{d \leftarrow not\ c; c \leftarrow not\ b\}$. Its unique TM is $\{c\}$ providing the global model $\{a, c\}$ together with the $\{a\}$ model for $a \leftarrow not\ a$.

5. Properties of the Tight Semantics

Forthwith, we prove some properties of TS, 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 5.1. ***Existence.*** *Every Normal Logic Program has a Tight Model.*

Proof. Let P be an NLP. $\widehat{P}$ is guaranteed to exist. So are MMs of any given NLP, in particular, for $\widehat{P}$ too. If $\widehat{P}$ has no *loops*, then every MM of $\widehat{P}$ is trivially Tight. In particular, if $\widehat{P}$ has no *loops* it means $\widehat{P}$ is stratified and the unique TM is its unique MM.

Consider now $\widehat{P}$ has *loops*, and that P_L is any such loop in $\widehat{P}$. Assume $\widehat{P}$ has no TMs. In this case, for every P_L there is no $M_L \subseteq M_{L_N}$ such that $Tight_{P:M_{L_N}}(M_{\overline{L}})$ holds. Since for every P_L it is always possible to compute an M_L and its respective M_{L_N}, the tightness condition must fail because $Tight_{P:M_{L_N}}(M_{\overline{L}})$ fails. But any P_L which does not depend on any other rule outside P_L is unaffected by any program division $P : M_{L'_N}$ where $M_{L'}$ is an MM of some other $P_{L'}$. Hence the hypothetical failure of tightness in holding of $M_{\overline{L}}$ in $P : M_{L_N}$ must be because all M_Ls of all P_L are not Tight in some $P_{L'''} = P_{L''} \cup P_L$ such that P_L depends on $P_{L''}$ and vice-versa. I.e., for all M_L of P_L, $Tight_{P_{L'''}:M_{L_N}}(M_{\overline{L}})$ must not hold. Since it is always possible to compute $M_{\overline{L}} = (M \setminus M_L) \cup \{M_L \cap heads(P_{L'''} : M_{L_N})\}$ it must be the case that for every $M_{L''}$ of each $P_{L''}$, $M_{L''} \cup M_L$ is not a consistent MM of $P_{L''} \cup P_L$, which is an absurdity because consistent MMs of any given program are always guaranteed to exist. ■

Theorem 5.2. ***Relevance of Tight Semantics.*** *The Tight Semantics is relevant.*

Proof. According to definition 3.1 a semantics Sem is relevant iff $a \in Sem(P) \Leftrightarrow a \in Sem(Rel_P(a))$ for all atoms a. Since the TS of a program P — $TS(P)$ — is the intersection of all its TMs, relevance becomes $a \in TS(P) \Leftrightarrow a \in TS(Rel_P(a))$ for TS.

$\Rightarrow$: We assume $a \in TS(P)$, so we can take any M such that $TM_P(M)$ holds, and conclude $a \in M$. Assuming, by contradiction, that $a \notin TS(Rel_P(a))$ then there is at least one TM of $Rel_P(a)$ where a is *false*. Let us write M_a to denote such TM of $Rel_P(a)$ where $a \notin M_a$. Since all TMs of P are MMs of $\widehat{P}$ we have two possibilities: 1) a is a fact in $\widehat{P}$ — in this case there is a rule (a fact) for a and hence this fact rule is in $Rel_P(a)$ forcing $a \in M_a$; 2) a is not a fact in $\widehat{P}$ — by definition of TM a can be in M only if a is the head of a rule and there is some MM $M_L \subseteq M$ of a loop $P_L \subseteq P$ such that $a \in M_L$. Since a must be the head of a rule in loop, that loop is, by definition, in $Rel_P(a)$. Since M is Tight in P, by definition so must be each and every of its subset MMs of loops; i.e., $a \in M_a$.

$\Leftarrow$: Assume $a \in TS(Rel_P(a))$. Take the whole $P \supseteq Rel_P(a)$. Again, a will be in every TM of P because a is in all TMs of $Rel_P(a)$, and, by definition, every TM of P always contains one TM of $Rel_P(a)$. ■

Theorem 5.3. ***Cumulativity of Tight Semantics.*** *The Tight Semantics is cumulative.*

Proof. By definition 4.1, the semantics of a program P is the intersection of its TMs. So, $a \in TS(P) \Leftrightarrow \forall_{TM_P(M)} a \in M$. For the TS semantics cumulativity becomes expressed by $\forall_{a,b}(a \in TS(P) \wedge b \in TS(P)) \Rightarrow a \in TS(P \cup \{b\})$

Let us assume $a \in TS(P) \wedge b \in TS(P)$. Since both $a \in TS(P)$ and $b \in TS(P)$, we know that whichever TM M and $M_L \subseteq M$ such that $a \in M_L$, $b \in TS(P : M_{L_N})$ holds; and in that case $P : M_{L_N} = (P \cup \{a\}) : M_{L_N}$. Hence, $b \in TS(P \cup \{a\})$. ■

Theorem 5.4. ***Stable Models Extension.*** *Any Stable Model is a TM 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 exactly the same step as the P/M one, but then only deletes the *not* b such that *not* $b \in M_N$. Moreover, the P/M division is performed using the whole M at once, whilst the $P : M$ considers not the whole M but only partial M_{L_N}s of M. Tightness requires consistency amongst the several individual M_{L_N}s whilst the $M = least(P/M)$ stability condition requires consistency throughout the whole M. We can thus say the division P/M performs all the steps the $P : M$ division does, and then some. In this sense the $M = least(P/M)$ stability condition demands from M all that Tightness does and even more. Hence, a model passing the stability condition is bound to be also a TM. ■

Theorem 5.5. ***Tight Semantics respects the Well-Founded Model.*** *Every Tight Model of P respects the Well-Founded Model of P — $\forall_{M:TM_P(M)} RespectWFM_P(M)$.*

Proof. Take any TM M of P. Since all TMs are MMs of $\widehat{P}$, M must contain all the facts of $\widehat{P}$, i.e., $M \supseteq WFM^+(P)$. Also MMs of $\widehat{P}$ are bound to be a subset of the heads of rules of $\widehat{P}$, hence $M \subseteq WFM^{+u}(P)$. ■

Due to lack of space, the complexity analysis of this semantics is left out of this paper. Nonetheless, a brief note is due. Tight Model existence is guaranteed for every NLP, whereas finding if there are any SMs for an NLP is NP-complete. Since TS 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^2_P-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 TS requires the computation of a minimal model — another NP-complete task. Hence the conjectured Σ^2_P-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 TS should have the complementary complexity of brave reasoning: co-Σ^2_P-complete.

One common objection to these kind of semantics concerns the notion of support. Tight models are not supported, considering the classical notion of support. However, we abide by a more general notion of support: an atom is 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.* 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, Tight model 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 TS, 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 or infinite chains), 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 TS 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 [Bar03].

A current avenue of further work already being taken follows the line of thought we laid out in [Per09] by partitioning an NLP into layers, a generalization of strata, to further segment the program and thus reduce the combinatorics of the Tightness test. Although not reducing the theoretical complexity class of the Tightness test, in practical implementations the syntactical partitioning of layering can have a substantial impact on performance.

References

[Bar03] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving.* Cambridge University Press, 2003.

[Bra01] 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.

[Dix95] J. Dix. A Classification-Theory of Semantics of Normal Logic Programs: I, II. *Fundamenta Informaticae*, XXII(3):227–255, 257–288, 1995.

[Fag94] F. Fages. Consistency of Clark's completion and existence of stable models. *Methods of Logic in Computer Science*, 1:51–60, 1994.

[Gel88] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, pp. 1070–1080. MIT Press, 1988.

[Gel91] 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.

[Leo02] 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.

[Lif92] Vladimir Lifschitz and Thomas Y. C. Woo. Answer sets in general nonmonotonic reasoning (preliminary report). In *KR*, pp. 603–614. 1992.

[Per05] L. M. Pereira and A. M. Pinto. Revised stable models - a semantics for logic programs. In G. Dias et al. (ed.), *Progress in AI*, *LNCS*, vol. 3808, pp. 29–42. Springer, 2005.

[Per09] L. M. Pereira and A. M. Pinto. Layer supported models of logic programs. In E. Erdem, F. Lin, and T. Schaub (eds.), *Procs. 10th LPNMR*, *LNAI*, vol. 5753, pp. 450–456. Springer, 2009. URL `http://centria.di.fct.unl.pt/$\sim$lmp/publications/online-papers/LSMs.pdf(longversion)`

[Syr01] Tommi Syrjänen and Ilkka Niemelä. The smodels system. In T. Eiter et al. (ed.), *LPNMR 2001*, *LNAI*, vol. 2173. Springer-Verlag, 2001.

[Tar72] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Computing*, 1(2):146–160, 1972.