Negotiating Logic Programs

Norman Foo and **Thomas Meyer**
National ICT Australia
and School of Computer Science and Engineering
University of New South Wales
NSW 2052, Australia

Yan Zhang and **Dongmo Zhang**
School of Computing and Information Technology
University of Western Sydney
Penrith South DC
NSW 1797, Australia.

Abstract

Negotiation between two agents is modelled as a one-time encounter between two extended logic programs. Each offers an answer set. Their mutual deal can be regarded as a trade on their answer sets. An agent can achieve this by weakening its program to surrender some literals that conflict with the offer of the other agent. We examine how ways of weakening affects the answer sets, and how they may be used to effect a deal. They are shown to satisfy the classificatios of outcomes that follow from known postulates for rational negotiation.

1 Introduction

Negotiation has been studied in many contexts (e.g. [Rosenschein and Zlotkin 94]. Ideally we would like to represent all kinds of intentions, attitudes, strategies, etc. of the agents who negotiate, but the complexity would be prohibitive for any kind of reasoning about them and their behavior. In game theory the usual models are greatly simplified in an attempt at rigorous analysis that can hopefully be extended to more realistic settings. Our work here can be regarded as a modest addition to recent research (e.g. [De Vos and Vermeir 02] [Foo et al. 04]) that introduced logic programs to reason about well-known game-theoretic scenarios. The novel features in this paper include: (1) there is no appeal to utilities; (2) agents are represented as general (extended) logic programs; (3) a negotiation is modelled as a trade between two agents with their respective program solutions as the commodity. A persuasive reason for adopting programs as models of agents is that the complexity of agent intentions, etc., can be gradually scaled up as the simpler models are understood. Two directions immediately suggest themselves: model the repeated encounter negotiations, and model agent epistemic states and intentions as rules. We first describe our agent model and briefly review the relevant logic programming background. Then methods for weakening a program, modelling partial concession by an agent in negotiation, are examined. A brief review of recent formal postulates on rational negotiation is next. Finally we evaluate our agent program concessions against these postulates.

2 Representation of Agents

While real-life negotiation between two agents is possibly iterative, involving repeated encounters in which the agents trade demands and concessions, for the moment we only model a single encounter in which both agents declare their demands simutaneously, and then each agent uses this mutual information to modify their original demands to achieve an agreement. The scenario in which an agreement is reached only after repeated encounters is akin to the notion of *extensive games*, while the alternative scenario of a one-time encounter is akin to that of a *normal form game* in the game theory context [Luce and Raiffa]. In that context it is known that there are translations between the two kinds of game formalisms, but normal form games are easier to analyse to understand the limits of what player strategies can achieve or gaurantee. Likewise, we confine our attention to single encounter negotiation to gain intuition that will be useful in our future work on repeated encounters.

We model each agent as an extended logic progam (ELP), a formalism introduced by Gelfond and Lifschitz [Gelfond and Lifschitz 90] to express both classical negation and negation-as-failure. Using ELPs permit us to make contradictory demands explicit and also admit incomplete knowledge by agents. The accepted semantics of ELPs are *answer sets* (op cit.), which are in turn based on the *stable model* semantics of a less expressive formalism — general or normal logic programs (NLP) — also introduced by Gelfond and Lifschitz [Gelfond and Lifschitz 88].

The one encounter model of negotiation is effectively the presentation by each agent of an answer set. Based on the pair of presented answer sets and assumptions about the predisposition (risky, cautious, obstinate, dominant, etc.) of the agents, they will arrive at the union of the modified answer sets as the agreement pair. Thus, the central issue in reaching this agreement is how each agent should modify its answer set. Since answer sets are determined by the agent program this modification is tantamount to changes in its program. Moreover, these changes should ideally be faithful to the intended modification of the answer set.

3 Review of Logic Programs

In this paper the logical language is propositional (ground atoms). A standard reference for the topics in this paragraph is the text by Lloyd [Lloyd 87]. The simplest logic

programs are the *definite logic programs (DLPs)* in which the rules are pure Horn clauses. A typical rule r is of the form $A \leftarrow B_1, \ldots, B_n$ where $A, B_1, \ldots, B_n$ are atoms. A definite program Π is a finite collection of such rules. The notation $head(r)$ means the atom in the consequent of rule r, and $body(r)$ means the set of atoms in its antecedent. Thus for the typical rule r above we have $head(r) = A$ and $body(r) = \{B_1, \ldots, B_n\}$, and r can therefore also be denoted by $head(r) \leftarrow body(r)$. the declarative semantics of Π is its smallest Herbrand model M_Π, and its (equivalent) operational semantics is its least fixed point $lfp(\Pi)$ or what amounts to the same thing the set of query (atoms) which succeed; since this is unique we may call it the least model of Π. By $Atoms(\Pi)$ we mean the set of atoms that occur in Π. A rule is *redundant* if it is never used to answer query B? for any atom $B \in lfp(\Pi)$. A more formal way to define redundancy is via the standard $T_\Pi \uparrow$ operator which records the rules used to produce $lfp(\Pi)$, but the informal view suffices here. The important fact is that redundant rules can be deleted with no effect on the semantics of the program. One easy consequence is this:

Lemma 1 *For a DLP Π if $A \notin lfp(\Pi)$ then any rule r with $head(r) = A$ is redundant.*

Default reasoning required the introduction of a kind of negation into logic programs, viz, inferring the negation of an atom if all attempts to demonstrate the truth of its positive form fails. This is the "negation-as-failure" notion of negation, denoted by $not\ A$, and in the *general logic programs (GLPs)* or *normal logic programs (NLPs)* such terms are allowed in the bodies of rules. We will call them normal logic programs (NLPs) for short. With this generalization the bodies of rules split into those terms that are atoms and those atoms that are negated using *not*; these sets are denoted respectively $pos(r)$ and $neg(r)$ for rule r. Thus a normal rule r can be written as $head(r) \leftarrow pos(r), not\ (neg(r))$. An example NLP rule r is $A \leftarrow B, C, not\ D, not\ E$, whence $head(r) = \{A\}$, $pos(r) = \{B, C\}$ and $neg(r) = \{D, E\}$. As before, $body(r)$ means $pos(r), not\ neg(r)$ and likewise with $Atoms(\Pi)$. The accepted semantics of NLPs is given by its Gelfond-Lifschitz transform [Gelfond and Lifschitz 88], which we now informally recall. Let S be a set of atoms and Π an NLP. Each rule r of Π is filtered by S as follows: if $neg(r) \cap S \neq \emptyset$ then r is blocked. Intuitively this is because S is considered to be a "guess" at a model of Π and hence its atoms are assumed to be true. Then no rule r with $A \in neg(r)$ and also $A \in S$ can be applicable since the "default" $not\ A$ is false relative to S. The surviving unblocked rules may still have negated atoms in their bodies, but by assumption these are automatically satisfied by S and therefore can be dropped from their bodies. Hence we end up with a subset of the original rules of Π in which the negated terms have been deleted, viz., we are back to a DLP. Since this program derived from Π is S-relative, it is customary to denote it by Π^S. Being a DLP Π^S has a least fixed point as its semantics. Gelfond and Lifshitz (op. cit.) define S to be a *stable model* of Π if $lfp(\Pi^S) = S$.

If we further generalize the syntax of NLP to allow the atoms to be classically negated, then each rule can have *literals* in its head and body, and also negated literals in its body. Such programs were introduced also by Gelfond and Lifschitz [Gelfond and Lifschitz 90] who called them *extended logic programs (ELPs)*. There are therefore two kinds of negation in ELPs — classical ($\neg$) and negation-as-failure (not). A typical ELP rule r is $\neg A \leftarrow B, \neg C, not\ D, not\ \neg E$. The models of ELPs can be regarded as stable models of an NLP in which the literals, say A and $\neg A$ are represented (or encoded) respectively by *atoms* A and A'. In fact this is essentially what Gelfond and Lifschitz did in their paper that introduced ELPs. The encoding of the preceding ELP rule r is the NLP rule r' = $A' \leftarrow B, C', not\ D, not\ E'$, an NLP rule. If this is done, then the original ELP Π is transformed to (more precisely, encoded by) a NLP Π', and the stable model semantics for NLP is applicable. Each consistent stable model M (no pair A and A' in it), when translated back to the original stable model literals (A to itself, A' to $\neg A$), is an *answer set* of Π.

4 DLP Reducts

Consider a definite logic program Π. Suppose for some subset $\Delta \subseteq M_\Pi$ of atoms we wish to modify Π to some definite program Π' with model $M_{\Pi'}$ such that $M_\Pi \setminus M_{\Pi'} = \Delta$, i.e., some atoms are lost in the new model.

The algorithm presented below removes one atom from the model of its input program; a generalization does this for sets of atoms. The single atom removal algorithm is a composition of two sub-algorithms, the first being the well-known *unfolding* of a program and the other a *retraction* of rules whose heads are atoms in Δ. Unfolding has been researched in detail since Tamaki and Sato [Tamaki and Sato 84] introduced it in the context of program optimization, and we refer to Aravindan and Dung [Aravindan and Dung 95] for the results needed for this sub-algorithm. To recall, given a definite program Π, its unfolding with respect to an atom A is a new program $Unfold(\Pi, A)$ obtained from Π by application of the following procedure:

1. For each pair of rules r and r' in Π with $head(r) = A$ and $A \in body(r')$, add the rule $head(r') \leftarrow body(r), body(r') - \{A\}$;
2. Delete rules r with $A \in body(r)$.

The intuitive idea behind unfolding with resepct to A is to eliminate A in the bodies of rules using partial execution. This works even if there are rules r in Π with $head(r) = A$ and $A \in body(r)$, i.e. there is a recursion in A (which are equivalent to the empty program [Lin 02] and are redundant) since they will be deleted in the second step. We can further simplify the new program by deleting other redundant rules in this manner: Remove all clauses r such that $B \in head(r)$ and $B \in body(r)$. We will assume that simplification is always done. Besides having no rule in $Unfold(\Pi, A)$ with A in its body, unfolding has a nice property: (see Dung, (op. cit.):

Observation 1 *Unfolding preserves the model of Π, i.e., the model of $Unfold(\Pi, A)$ is the same.*

The retraction part is this procedure.

> Given a definite program Π for which every rule r satisfies $A \notin body(r)$, the retract of atom A from

Π is a new program $Retract(\Pi, A)$ obtained from Π by deleting all rules r with $head(r) = A$.

Observe that $A \notin Atoms(Retract(\Pi, A))$. The retract of an atom A from a program is undefined if the condition $A \notin body(r)$ is violated for some r in it.

Given a definite program Π with model M the following transformation produces a new definite program $Reduce(\Pi, A)$ whose model is $M \setminus \{A\}$, i.e., eliminates the atom A from the model M.

$$\Pi' = Unfold(\Pi, A);$$
$$Reduce(\Pi, A) = Retract(\Pi', A)$$

Retracting an atom is weaker than the operation of *contraction* in the well-known AGM theory of belief revision [Gardenfors 88]. In the AGM framework the contraction of an atom A from a theory T may also result in "side-effects" of contractions of other atoms from T. However, the retract here is more like its namesake in Prolog, and removes only rules with that atom in their heads.

Lemma 2 *The model of* $Reduce(\Pi, A)$ *is* $M_\Pi \setminus \{A\}$.

Lemma 3 *The models of* $Reduce(Reduce(\Pi, A), B)$ *and* $Reduce(Reduce(\Pi, B), A)$ *are the same.*

The proof of this is by brute force, showing the equality (after simplification) of the two orders of reduction.

If we are only interested in the semantics of the transformed program, Lemma 3 justifies the use of $Reduce(\Pi, \Delta)$ to denote any of the equivalent programs that result from the algorithm when applied successively to Π using the atoms from Δ in any order. These lemmas imply the following proposition.

Proposition 1 *The model of* $Reduce(\Pi, \Delta)$ *is* $M_\Pi \setminus \Delta$.

5 NLP and ELP Reducts

As mentioned in section 2 above agents will be represented as ELPs. Generally these do not have unique answer sets. So any reduction algorithm for them has to be a generalization of that for DLPs. The diagram below summarizes our approach. On the left hand side the objects are NLPs (ELPs) under the components of the reduction algorithm. On the right hand side the objects are DLPs. As indicated in the diagram the DLPs on the right are obtained from a corresponding NLP (ELP) via a Gelfond-Lifschitz transformation (GLT) relative to a stable model (answer set) S. The diagram is generic for stable models (answer sets) S, i.e. there is one for each such set. The idea is to refer questions about the stable models (answer sets) on the left to the unique models (least fixed points) on the right. By the remarks in section 3 it suffices to examine NLPs and stable models. The questions for ELPs can be decided via the NLP encoding of an ELP. The reduction algorithm for NLP also consists of two steps, of which the first is unfolding. This is virtually the same as that for DLPs where unfolding of A in the body of a rule is simply the syntactic replacement of A by the body of a rule with A in its head. That this body may contain positive and negated literals is not material. More precisely, if r is $A \leftarrow pos(r), neg(r)$ and r' is $B \leftarrow pos(r'), neg(r')$ then an added rule is $B \leftarrow pos(r') \setminus \{A\}, pos(r), neg(r), neg(r')$.

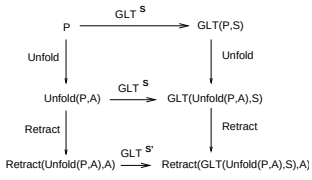

Figure 1: Gelfond-Lifschitz commutation with reduction

Observation 2 *Unfolding NLPs and DLPs is known [Aravindan and Dung 95] to preserve models, so the commutativity of the upper rectange of figure 1 follows.*

The commutativity of the lower rectangle is about the retract step. Certainly we still retract all rules r with $head(r) = A$, as well as those with $A \in pos(r)$. The new consideration is what we do with a rule r with $A \in neg(r)$. To justify the two ways of treating this we make a digression into two interpretations of negation-as-failure.

5.1 Negation as Failure

There are two ways of treating the retraction of a negation-as-failure atom in a rule. We illustrate them using an an example program (call it Π) adapted from [Gelfond and Lifschitz 90]. $\{Eligble(X) \leftarrow HighGPA(X); Eligible(X) \leftarrow FairGPA(X), not\ Advantaged(X); Advantaged(X) \leftarrow Wealthy(X); Wealthy(a); HighGPA(b); FairGPA(c).\}$ The informal idea of these rules is to determine eligibility of college applicants for scholarships. $Advantaged(X)$ signifies that X comes from an advantaged background, so the second rule captures the idea that disadvantaged applicants with only a fair GPA are still eligible.

The only stable model of Π is S = $\{Wealthy(a), HighGPA(b), FairGPA(c), Advantaged(a), Eligible(b), Eligible(c)\}$, which captures the desired outcome.

Suppose we weaken Π by supressing the atom $Advantaged(\_)$. Unfolding of Π with respect to this atom has no effect on it. If we now delete the rule with $Advantage(X)$ in the head the resulting program Π' is: $\{Eligble(X) \leftarrow HighGPA(X); Eligible(X) \leftarrow FairGPA(X), not\ Advantaged(X); Wealthy(a); HighGPA(b); FairGPA(c).\}$ There are two ways to regard $not\ Advantaged(X)$ in the rule of Π'. One way is to simply suppress it, giving the program Π'': $\{Eligble(X) \leftarrow HighGPA(X); Eligible(X) \leftarrow FairGPA(X); Wealthy(a); HighGPA(b); FairGPA(c)\}$. Π'' has one stable model S'' = $\{Wealthy(a), HighGPA(b), FairGPA(c), Eligible(b), Eligible(c)\}$, which has lost the atom $Advantaged(a)$ from the original stable model.

Another way is to delete the rule in which $not\ Advantaged(X)$ occurs, giving Π''': $\{Eligble(X) \leftarrow HighGPA(X); Wealthy(a); HighGPA(b); FairGPA(c)\}$. Π''' has one stable model S''' = $\{Wealthy(a), HighGPA(b), FairGPA(c), Eligible(b)\}$, which has lost both $Advantaged(a)$ and $Eligible(c)$ from the original model. Intuitively, in the absence of information about $Advantaged(c)$, S'' still yields a desired outcome while

S''' does not[1]. The informal reason is that the occurrence of $not\ Advantaged(X)$ in the orginal rule is intuitively interpreted as a *default test* in the context of fair GPAs.

Contrast this with an example rule attributed to John McCarthy: $Cross \leftarrow not\ Train$, presumably occurring in a program that decides when a car should cross a level-crossing. The retracting of $Train$ by just deleting it in the rule would result in the fact $Cross \leftarrow$, which may be dangerous. In this case the intuitive treatment is to *delete* the rule altogether. The difference between this use of negation-as-failure and the preceding one is that here it is a *support* for the head of the rule.

Hence, both ways of treating retract should be considered when the informal semantics is not available.

5.2 Retracting negated literals

The intuitions of subsection 5.1 above suggest two ways of handling the $not\ A$ terms in the bodies of rules at the retract step, viz., (1) delete $not\ A$ in rules, and (2) delete the rules whose bodies contain $not\ A$. It is natural to call the former *weak* and the latter *strong*. Therefore, for NLPs we have two versions of the retraction algorithm:

> *Weak retract:*
> Given a definite program Π for which every rule r satisfies $A \notin pos(r)$, the retract of atom A from Π is a new program $WRetract(\Pi, A)$ obtained from Π by deleting all rules r with $head(r) = A$, and by deleting $not\ A$ from all rules.
>
> *Stong retract:*
> Given a definite program Π for which every rule r satisfies $A \notin pos(r)$, the retract of atom A from Π is a new program $SRetract(\Pi, A)$ obtained from Π by deleting all rules r with $head(r) = A$, and by deleting all rules whose bodies contain $not\ A$.

The two overall versions of reduction for NLPs are then:
Program $WReduce(\Pi, A)$ is obtained from Π by:

$$\Pi' = Unfold(\Pi, A);$$
$$WReduce(\Pi, A) = WRetract(\Pi', A)$$

Program $SReduce(\Pi, A)$ is obtained from Π by:

$$\Pi' = Unfold(\Pi, A);$$
$$SReduce(\Pi, A) = SRetract(\Pi', A)$$

Analogous to the case of DLPs, both these reductions remove A from *all* stable models. To show this we refer to the commutative diagram in figure 1. The following observation is a direct consequence of the defintion of stable models.

Observation 3 *At any horizontal level in figure 1, S is a stable model of the program on the left if and only if it is the least fixed point (or least model) of the program on the right.*

This observation reduces questions about a stable model S of an NLP Π to questions about the least fixed points of the DLP Π^S (= $GLT(\Pi, S)$ in the figure), and thence they can be answered using results from section 4. The next observation follows from the fact that there are no not terms in DLPs.

[1]However, there is a scenario in which S''' is reasonable, viz., when it is intended to repeal affirmative action. This is analogous to the Train example.

Observation 4 *If Π is a DLP then $WRetract(\Pi, A)$ = $SRetract(\Pi, A)$ = $Retract(\Pi, A)$.*

Since NLP can have more than one stable model, the analog of lemma 2 is something along the lines of: Suppose S is a stable model of Π; if $S \setminus \{A\}$ is non-empty it is a stable model of $WSReduce(\Pi, A)$, where WS is either W (weak) or S (strong). However, as suggested by the following example, it is slightly more complicated. Consider the program Γ: $\{A \leftarrow not\ E; C \leftarrow A, not\ D; E \leftarrow not\ A\}$ The stable models of Γ are $\{E\}$ and $\{A, C\}$. Deleting A from them yields $\{E\}$ and $\{C\}$. On the other hand, if we do weak reduction the only stable model is $\{E\}$, while strong reduction yields the stable model $\{C\}$.

Proposition 2 *Suppose S is a stable model of Π and $S \setminus \{A\}$ is non-empty.*

1 *If $A \notin S$ then S is also a stable model of $WReduce(\Pi, A)$, but not always of $SReduce(\Pi, A)$.*

2 *If $A \in S$ then $S \setminus \{A\}$ is a stable model of $SReduce(\Pi, A)$, but not always of $WReduce(\Pi, A)$.*

Let $Mod(\Pi)$ be the set of stable models of Π. The generalization of lemma 3 is:

Lemma 4

$$Mod(WReduce(WReduce(\Pi, A), B)) = Mod(WReduce(WReduce(\Pi, B), A))$$
$$Mod(SReduce(SReduce(\Pi, A), B)) = Mod(SReduce(SReduce(\Pi, B), A))$$

Therefore for a set Δ we have unambiguous meanings for $WReduce(\Pi, \Delta)$ and $SReduce(\Pi, \Delta)$. The following is then a consequence of proposition 2.

Proposition 3 *Suppose S is a stable model of Π and $S \setminus \Delta$ is non-empty.*

1 *If $\Delta \cap S = \emptyset$ then S is also a stable model of $WReduce(\Pi, \Delta)$, but not always of $SReduce(\Pi, \Delta)$.*

2 *If $\Delta \subset S$ then $S \setminus \Delta$ is a stable model of $SReduce(\Pi, \Delta)$, but not always of $WReduce(\Pi, \Delta)$.*

Corollary 1 *$A \notin S$ for each $S \in Mod(WReduce(\Pi, A))$. $A \notin S$ for each $S \in Mod(SReduce(\Pi, A))$.*

Definition 1 *Fix a language. An NLP (ELP) program Π' is a weakening of an NLP (ELP) program Π if there is a one-to-one pairing between their models, such that for each $M' \in Mod(\Pi')$ there is an $M \in Mod(\Pi)$ such that $M' \subseteq M$.*

In weakening a program, literals are lost from its stable models. In the negotiation context, an agent (program) may choose to surrender a particular subset Δ of its literals to avoid a conflict with the other party. The weakenings that result in losing no more than is necessary are *maximal*. Formally a weakening Π' of Π is *Δ-maximal* if for each model pair (M', M) it is the case that $M \setminus M' \subseteq \Delta$.

Corollary 2 *$SReduce(\Pi, \Delta)$ is Δ-maximal when $\Delta \subset S$ for each $S \in Mod(\Pi)$.*

6 Negotiation between Programs

Imagine an encounter between two agents 1 and 2 represented respectively as ELPs (NLPs) Π_1 and Π_2. It is natural to regard any negotiated agreement or deal between them to have a connection with the answer sets (stable models) of the two programs. One way to do this is to consider the "merged" program $\Pi_1 \cup \Pi_2$ and its answer sets. We might then proceed to perform the reduction algorithm on this program, nominating subsets of literals to retract to achieve modified answer sets according to different rationality criteria. We consider this approach in a companion paper[2], but did not evaluate its behavior relative to rationality. Here we examine an alternative and more concrete approach by regarding the encounter as these successive events: Agents 1 and 2 each choose one of their answer sets, reveal the literals in the sets as their demands, and then each modifies its demands to arrive at an agreement. The modification is achieved by reductions on the agent program. Reductions effect the concession by agents of parts of their demands. This is evaluated for rationality using the postulates in the (next) subsection 6.1.

We are very much aware of the simplicity of our formalization of agent negotiation as a one time encounter. One does not have to reflect much to see that real-life negotiation involves multiple encounters. What passes for bargaining is in fact repeated encounters in which the agents weaken some demands, accept some of their opponent's demands, but come back with new demands, etc. until mutual satisfaction is reached. In the ELP context weakening can be captured as reduction, acceptance can be interpreted in two ways, and new demands may amount to switching answer sets. Acceptance of some demands of the other agent can be either *passive* or *active*. The passive mode just blocks the agent from ever demanding literals contradicting the accepted set. The active mode is *program expansion* by adding the accepted literals as new facts into the agent's program. This is like Prolog's assert meta-procedure, and the dual to retract. A side effect of this active mode is the alteration of the program's answer sets. Answer set switching can also be caused by another aspect of real-life negotiation that we have ignored, viz., preference orderings on rules and literals. These interesting features are being investigated.

6.1 Postulates for Rational Negotiation

We review recent work by Meyer, et al. [Meyer et al. 04] on rationality postulates that try to capture how rational agents should behave in a one encounter negotiation. These postulates are expressed abstractly in logic. We will evaluate our paricular logic programming models of agents and algorithms for program change against these postulates so that we have some guarantees of rationality. The review below is essentially a summary of the relevant section of the cited paper (op. cit.). The language $\mathcal{L}$ of these postulates is is finitely generated and propositional. Falsum is denoted by $\bot$, logical entailment by $\models$, logical equivalence by $\equiv$ and logical closure by Cn. A *theory* is a set of sentences closed under logical entailment. $M(K)$ denotes the *models* of a set of sentences K and $M(\alpha)$ that of a single sentence α.

[2]Details and identities suppressed to maintain anonymity.

A deal D is defined as an abstract object. Any deal is defined with respect to a *demand pair* $K = (K_1, K_2)$, with K_i $(i = 1, 2)$, being a *consistent* theory of $\mathcal{L}$, representing the initial demands, or demand set, of agent i. For simplicity agents are assumed to be logically omniscient. The theory developed models as being concerned with the *outcome* of the process of negotiation rather than the process by which this is reached. This is close to the one-time encounter model we choose for this paper and to normal form games. The outcome $O(D)$ of a deal D is formalized as a set of sentences representing the demands which both agents have agreed upon. A deal D is *outcome-permissible* iff $O(D)$ satisfies the following rationality postulates:

(O1) $O(D) = Cn(O(D))$

(O2) $O(D) \not\models \bot$

(O3) If $K_1 \cup K_2 \not\models \bot$ then $O(D) = Cn(K_1 \cup K_2)$

(O4) If $O(D) \cup (K_1 \cap K_2 \not\models \bot$ then $(K_1 \cap K_2) \subseteq O(D)$.

(O1) ensures that outcomes are theories and (O2) that outcomes are consistent. (O3) states that if the initial demand sets do not conflict, an agent will accept all demands of the other agent, i.e. agents cooperate when possible. (O4) says that if the outcome and common demands are consistent, then all common demands have to be included. The intuitive justification is that if a potential outcome O is *consistent* with the demands that the agents have in common, it would be to the benefit of both rather to strengthen O to contain *all* commonly held demands.

6.2 A classification of deals

The constraints placed on outcomes by (O1)-(O4) lead naturally to a taxonomy of deals in which we distinguish between four kinds of deals. A *trivial* deal is one for which the outcome is $Cn(K_1 \cup K_2)$. This can only occur when the combination of the initial demand sets is consistent. In the next type of deal one of the agents, the *master*, gets to keep all its demands. An *i-dominant* deal ($i = 1, 2$) is one in which agent i plays the role of the master. Observe that the different supersets of the demands of agent i correspond to the different i-dominant deals. The third type of deal is the class of *cooperative* deals, where the outcome is consistent with the initial demand set of each agent and includes their common demands. Finally, we have the class of *neutral* deals in which the outcome of negotiation is inconsistent with the common demands. Meyer, et al. (op cit.) defined these types of deals formally and proved that for any negotiation scheme that satisfies the above rationality postulates every deal is one of these types. For our purpose here however, the preceding informal description suffices to classify negotiation between two agents represented as NLPs (and hence as ELPs).

6.3 Algorithm for a Deal

The notation $[\neg]L$ means (1) $\neg L$ when L is an atom, and (2) A when L is the negation $\neg A$ of atom A. Extending the notation, by $[\neg]\Delta$ where Δ is a set of literals is meant the set $\{[\neg]L \mid L \in \Delta\}$. Suppose agents 1 and 2 declare respective answer sets M_1 and M_2 as their demands. The contradiction set $Contrad_1$ of 1 is the set $\{L \in M_1 | [\neg]L \in M_2\}$, and symmetrically for $Contra_2$; thus $Contra_1 = [\neg]Contrad_2$

and symmetrically with indices swapped. The initial agreement set $Agree$ of both 1 and 2 is $M_1 \cap M_2$. An agent 1 may wish to keep a set $\Psi_1 \subseteq M_1$, giving up $M_1 \setminus \Psi_1$. The subset $\Psi_1 \cap Contrad_1$ has the literals that conflict with some in M_2, so agent 2 will have to give up at least $[\neg](\Psi_1 \cap Contrad_1)$ for consistency. If the new answer sets are to be consistent with each other, and both keep the agreement set, then the choice of Ψ_1 and Ψ_2 are constrained as follows: (1) $Agree \subseteq \Psi_1 \subseteq M_1 \cup (M_2 \setminus Contrad_2)$ and $Agree \subseteq \Psi_2 \subseteq M_2 \cup (M_1 \setminus Contrad_1)$ (2) $\Psi_1 \cap Contrad_1 \neq \emptyset$ implies $[\neg](\Psi_1 \cap Contrad_2) \cap \Psi_2 = \emptyset$ and $[\neg](\Psi_2 \cap Contrad_1) \cap \Psi_1 = \emptyset$. Call any pair (Ψ_1, Ψ_2) that satisfies the constraints *admissible*.

Deal Algorithm:

For any admissible pair (Ψ_1, Ψ_2):

$$\Pi_1' = SReduce(\Pi_1, M_1 \setminus \Psi_1)$$
$$\Pi_2' = SReduce(\Pi_1, M_1 \setminus \Psi_2)$$
$$M_1' = \Psi_1;\ M_2' = \Psi_2$$
$$Outcome = M_1' \cup M_2'$$

By corollary 2 the algorithm results in new programs Π_1' and Π_2' with new answer sets M_1' and M_2'[3]. Some further easily verified observations on the algorithm:

- If $Contrad_1 = \emptyset$ (and therefore also $Contrad_2 = \emptyset$) then $\Psi_1 = M_1$ and $\Psi_2 = M_2$ is an admissible pair. In that case the $SReduce$ function in the algorithm returns the original programs. The outcome is then simply the union of the two original answer sets of the agents, viz., the *trivial* deal.
- If $M_1 \subseteq \Psi_1$ then by (2) any Ψ_2 such that $\Psi_2 \cap Contrad_2 = \emptyset$ is an admissible pair. For such a pair the outcome is a *1-dominant* deal. The symmetric case is a *2-dominant* deal.
- If $\Psi_1 = M_1 \setminus Contrad_1$ and $\Psi_2 = M_2 \setminus Contrad_2$, the outcome is a *cooperative* deal.

These conform to the classification of outcome types in subsection 6.2 above. The apparently missing one is the *neutral* outcome. We conjecture that such outcomes will manifest themselves once preferences and fact assertions are introduced. However, an arguable view is the following. The outcome above is only ostensibly cooperative if we consider M_1 and M_2 as *sets*. If we admit logical closure Cn, the picture changes. Here is an example. Suppose $M_1 = \{p, q, r\}$ and $M_2 = \{\neg p, \neg q, r\}$. Then set-theoretically we have $M_1 \cap M_2 = \{r\}$; but if we had considered instead $Cn(M_1)$ and $Cn(M_2)$ the formula also common to both is $p \leftrightarrow q$. Hence if consequence closure Cn is admitted as part of the description of stable models, an outcome in which the agents 1 and 2 give up $\{p, q\}$ and $\{\neg p, \neg q\}$ respectively is *not* a cooperative deal since it fails to preserve the common (implicit) formula $p \leftrightarrow q$; in fact this is a *neutral* outcome. Evidently, we have to be careful in such evaluations against rationality postulates whose theorems assume logical closures. We are currently investigating preferences, assertions and logical closure to shed light on these issues.

[3] A case can be made that it is the programs that should be regarded as the outcomes, and their stable models are extensional side effects.

References

[Aravindan and Dung 95] C. Aravindan and P.M. Dung, "On the Correctness of Unfold/Fold Transformation of Normal and Extended Logic Programs", J. Log. Program, 24(3): 201-217, 1995.

[De Vos and Vermeir 02] M. De Vos and D. Vermeir, "Dynamic Decision Making in Logic Programming and Game Theory", in B McKay and J Slaney (Eds) AI02: Advances in Artificial Intelligence, 15th Australian Joint Conference on Artificial Intelligence, Springer LNAI 2557, pp. 36-57, 2002.

[Foo et al. 04] N. Foo, T. Meyer and G. Brewka, "LPOD Answer Sets and Nash Equilibria", in M. Maher (ed), Advances in Computer Science, ASIAN 2004, 9th Asian Computer Science Conference, Springer LNCS 3321, pp. 343-351, 2004.

[Gardenfors 88] P. Gardenfors, *Knowledge In Flux*, MIT Press, Cambridge, MA, 1988.

[Gelfond and Lifschitz 88] M. Gelfond and V.Lifscitz, "The Stable Model Semantics for Logic Programming", Proceedings of the Fifth International Conference on Logic Programming, pp 1070-1080; R. Kowalski and K. Bowen (eds), MIT Press, 1988.

[Gelfond and Lifschitz 90] M. Gelfond and V.Lifscitz, "Logic programs with classical negation", Proceedings of the 7th International Conference on Logic Programming, pp 579-597, MIT Press 1990.

[Lin 02] F. Lin, "Reducing strong equivalence of logic programs to entailment in classical propositional logic", Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning, KR'02, pp 170-179, Morgan Kaufmann Publishers, 2002.

[Lloyd 87] J. Lloyd, *Foundations of Logic Programming*, 2nd ed., Springer-Verlag, New York, 1987.

[Luce and Raiffa] D.R. Luce and H. Raiffa, *Games and Decisions: Introduction and Critical Survey*, John Wiley, 1957.

[Meyer et al. 04] T. Meyer, N. Foo, D. Zhang and R. Kwok, "Logical Foundations of Negotiation: Outcome, Concession and Adaptation", Proceedings the Nineteenth National Conference on Artificial Intelligence, AAAI'04, pp 293-298, 2004.

[Rosenschein and Zlotkin 94] J. Rosenschein and G. Zlotkin, *Rules of Encounter*, MIT Press, MA., 1994.

[Tamaki and Sato 84] H. Tamaki and T. Sato, "Unfold/fold transformations of logic programs", Proceedings of the Second International Conference on Logic Programming, pp 127-138, Uppsala University, 1984.