Set Based Logic Programming

H.A. Blair[1], V.W. Marek[2], J.B. Remmel[3], and A. Rivera[1]

[1] EECS Department, Syracuse University, Syracuse, NY 13244
[2] CS Department, University of Kentucky, Lexington, KY 40506
[3] Mathematics Department, University of California at San Diego, La Jolla, CA 92903

Abstract. In a previous paper, the authors showed that the mechanism underlying Logic Programming can be extended to handle the situation where the atoms are interpreted as subsets of a given space. In a such situation, the atoms of the underlying language corresponding to a logic program P are interpreted as sets, the one step consequence operator applied to a set S is interpreted as the union of all sets corresponding to atoms which are the heads of clauses whose body is satisfied by S, and the models of the program are interpreted as subsets of the space. It turns out that the operator approach to Logic Programming can be transferred to such situations. The concepts of supported and stable models of programs also naturally transfer. In this paper, we show that this set based formalism for Logic Programming naturally supports a variety of options. For example, if the underlying space has a topology, one can insist that the basic one-step consequence operator always produces a closed set or always produce an open set. We develop a general framework for set based programming involving monotone *idempotent* operators and demonstrate the utility of this approach by giving a spatial logic program representing two cooperating agents in a continuous environment.

1 Introduction

In [BMR01], the authors developed an extension of the logic programming paradigm which can directly reason about regions in space and time as might be required, for example, for applications in graphics, image compression, or job scheduling. Thus instead having the intended underlying universe be the Herbrand base of the program, one replaces the underlying Herbrand universe by some fixed space X and has the atoms of the program specify subsets of X, i.e. elements of the set 2^X, the set of all subsets of X.

If we reflect for a moment on the basic aspects of logic programming with an Herbrand model interpretation, a slight change in our point of view shows that interpreting atoms as subsets of the Herbrand base is a natural thing to do. In normal logic programming, we determine the truth value of an atom p in an Herbrand interpretation I by declaring $I \models p$ if and only if $p \in I$. However, this is equivalent to defining the *sense*, $\sigma(P)$, of a ground atom p to be the set $\{p\}$ and declaring that $I \models p$ if and only if $\sigma(p) \subseteq I$. By this simple move, we have

Dagstuhl Seminar Proceedings 05171
Nonmonotonic Reasoning, Answer Set Programming and Constraints
http://drops.dagstuhl.de/opus/volltexte/2005/266

permitted ourselves to interpret the sense of an atom as a subset, rather than the literal atom itself.

This given, we showed in [BMR01] that it is a natural step to take the *sense* $\sigma(p)$ of ground atom p to be a fixed assigned subset of some nonempty set X and to define a $I \subseteq X$ to be a model of p, written $I \models P$, if and only if $\sigma(p) \subseteq I$. This type of model theoretic semantics makes available, in a natural way, multiple truth values, intensional constructs, and interpreted relationships among the elements and subsets of X. Observe that the assignment σ of a *sense* to ground atoms is intrinsically intensional. Interpreted relationships among the elements and subsets of X allow the programs that use this approach, which was called *spatial logic programing* in [BMR01], to serve as front-ends for existing systems and still have a seamless model-theoretic semantics for the system as a whole.

It turns out that if the underlying space X has structure such as a topology or an algebraic structure such as a group, ring, field or vector space, then a number of natural options present themselves. For example, if we are dealing with a topological space, one can compose the one step consequence operator T with an operator that produces, e.g. topological closures of sets or interiors of sets. In such a situation, one ensures that the T always produces closed sets, or always produces open sets. Similarly, if the underling space is a vector space, one might insist that T always produces a subspace, or perhaps a convex closure. Notice that each of the operators: *closure*, *interior*, *span* and *convex-closure* are monotone *idempotent* (i.e. $op(op(I)) = op(I)$) operators. We call such an operator a *miop* (pronounced "my op").

One also has a variety of options for how to interpret negation. In normal logic programming, a model M satisfies $\neg p$ if $p \notin M$. From the set-based point of view when p is interpreted as a singleton $\{p\}$, this would be equivalent to saying that M satisfies $\neg p$ if (i) $\{p\} \cap M = \emptyset$, or (equivalently) (ii) $\{p\} \not\subseteq M$. When the sense of p is a set with more than one element it is easy to see that saying that M satisfies $\neg p$ if $\sigma(p) \cap M = \emptyset$ (a strong negation) is different from saying that M satisfies $\neg p$ if $\sigma(p) \not\subseteq M$ (a weaker negation). There are thus two natural interpretations of the negation symbol. Again, when the underlying space has structure, one can get even more subsidiary types of negation by taking M to satisfy $\neg p$ if $cl(\sigma(p)) \cap M = cl(\emptyset)$, or by taking M to satisfy $\neg p$ if $cl(\sigma(p)) \not\subseteq M$ where cl is some natural miop.

The main goal to this paper is extend spatial logic programming paradigm logic programming of [BMR01] to a full set based logic program paradigm with associated miops. The outline of this paper is as follows. In sections 2 and 3, we shall briefly review the spatial logic programming paradigm as given in [BMR01]. In section 4, we shall formulate a general set based logic programming formalism when the underlying space has natural miops. We shall give several examples where the same program can give different results depending on which miop and/or negation operator we use. Finally, in section 5, we shall give an application of our formalism to show how one can represent and reason about the coordination of agents in a continuous space.

2 Spatial Logic Programs: syntax and semantics

Before giving the general definitions of our formalism for set based logic programming with miop operators, we shall first recall the definitions of spatial logic programs as developed in [BMR01].

The syntax of spatial logic programs is based on the syntax of the formulas of what we define as *spatially augmented first-order logic.* Spatial augmentation is an intensional notion. The syntax of spatial programs will essentially be the syntax of DATALOG programs with negation, but augmented by certain *intensional connectives* such as union and intersection that are designed to make programming in a spatial logic programming setting easier.

The use of intensional connectives allows for operations on what we call the senses of ground atoms described in the next section to materially contribute to determining the models of programs. The expressive power of intensional connectives allows us to capture functions and relations intrinsic to the domain of a spatial logic program, but independent of the program. It is this feature that permits spatial logic programs to seamlessly serve as front-ends to other systems. Intensional connectives correspond to back-end procedures and functions. However, it turns out that intensional connectives can be eliminated from programs by using miops. The trade-off is a matter of expressive convenience and naturalness.

Definition 1. A **spatially augmented first-order language** (**spatial language**, for short) $\mathcal{L}$ is a quadruple $(L, X, \sigma, \mathcal{I})$, where
1) L is a language for first-order predicate logic without function symbols other than constants,
2) X is a nonempty (possibly infinite) set, called the **interpretation space**,
3) σ is a mapping from the ground atoms of L to the power set of X, and
4) $\mathcal{I}$ is a possibly infinite alphabet of symbols called **intensional** connectives. The collection is required to contain logical intensional connectives, corresponding to the union, intersection, and complement operators on 2^X as well as the constant unary operator that returns X. Each intensional connective is equipped with a fixed interpretation as an operator of some finite arity on 2^X.

Although $\mathcal{L}$ may have infinitely many intensional connectives, we will assume that in any spatial logic program P contains only finitely many such intensional connectives.

The mapping σ, the interpretation space X, and the interpretations of the intensional connectives might seem to properly belong in the semantics of spatially augmented languages. However, these languages are to be thought of as having a fixed partial interpretation, and hence the interpretation space, sense assignment, and the interpretations of the intensional connectives should be fixed by the language analogously to fixing the interpretation of the equality symbol in ordinary first-order languages as the identity relation.

We now define the *intensional atoms* of $\mathcal{L}$ in the usual inductive manner.

Definition 2. 1) An atomic formula A of L, the underlying first-order language component of $\mathcal{L}$, is an intensional atom, which we call a *primitive* atom. The predicate symbol of A is the *principal functor* of A and
2) If $\varphi_1, \ldots, \varphi_n$ are intensional atoms and η is an n-ary intensional connective, then $\eta(\varphi_1, \ldots, \varphi_n)$ is an intensional atom, whose principal functor is η.

The remaining intensional formulas of $\mathcal{L}$ are built up from intensional atoms in the usual way. It should be noted that intersection is *not* representable as familiar Boolean connectives. This will become clear after we present the semantics. We can then extend the notion of sense to arbitrary intensional ground atoms inductively by declaring that the sense of intensional ground atom $\eta(\varphi_1, \ldots, \varphi_n)$ to be given by

$$\sigma(\eta(\varphi_1, \ldots, \varphi_n)) = f(\sigma(\varphi_1), \ldots, \sigma(\varphi_n))$$

where the interpretation of η is a function $f : (2^X)^n \longrightarrow 2^X$.

We now define the class of spatial logic programs of the spatial language $\mathcal{L}$.

Definition 3. A **spatial logic program** has three components.
1) The language $\mathcal{L}$ which includes the interpretation space and the sense assignment.
2) The IDB (**Intentional Database**): A finite set of program clauses, each of the form $A \leftarrow L_1, \ldots, L_n$, where each L_i is a *literal*, i.e. an intensional atom or the negation of an intensional atom, and A is an intensional atom.
3) The EDB (**Extensional Database**): A finite set of intensional ground atoms.

Given a spatial logic program P, the *Herbrand base* of P is the Herbrand base of the smallest spatial language over which P is a spatial logic program.

For the rest of this section, we shall assume that the classes of spatial logic programs that we consider always are over a language for first-order logic L with no function symbols excepts constants, a fixed set X and a set of intensional connectives.

Informally, we think of the Herbrand universe Λ_L of the underlying language L, i.e. the set of constant symbols of L, as being a set of indices which we may employ to suit whatever purpose is at hand. We let HB_L denote the Herbrand base of L, i.e. the set of ground intensional atoms of L. We omit the subscript L when the context is clear. Let X be a nonempty set, 2^X the powerset of X, and let $\sigma : \mathrm{HB}_L \longrightarrow 2^X$. The subset of X, $\sigma(p)$, is called the **sense** of the ground atom p (with respect to X). An **interpretation** I of the spatial language $\mathcal{L} = (L, X, \sigma, \mathcal{I})$ is a subset of X. A ground intensional atom p is satisfied by the interpretation I, with respect to sense assignment σ (denoted by $I \models_\sigma p$) if and only if $\sigma(p) \subseteq I$. After introducing miops we will modify the $\models$ relation.

We note that sense assignments σ can be used to partition the ground atoms into multiple sorts. For example, let X be the disjoint union of X_1 and X_2. Let HB_L be the disjoint union of A_1 and A_2, and choose σ such that $\sigma(p) \subseteq X_i$ for $p \in A_i$, $i = 1, 2$.

The preceding definition allows us to extend the satisfaction relation to all intensional formulas with respect to 2-valued logic in the usual way. We could similarly define truth-valuations from subsets of X together with ground atoms into larger sets of truth values.

We now extend the the satisfaction relation to arbitrary formulas. Because of the diversity of notions of negation available, we will employ a mapping α_I corresponding to each $I \subseteq X$ from the set of sentences, i.e. the set of all formulas without free occurrences of variables, to three truth values $\mathbf{t}$, $\mathbf{f}$, and $\perp$. We first define α_I on the ground intensional literals, i.e. ground intensional atoms and their negations. α_I is more interesting when extended to all sentences.

We are assuming that the satisfaction relation $I \models_\sigma A$ on ground intensional literals A has been given, i.e., for each ground intensional atom A,

$$\alpha_I(A) = \begin{cases} \mathbf{t} & \text{if } I \models A \\ \mathbf{f} & \text{if } I \models \neg A \\ \perp & \text{otherwise.} \end{cases}$$

Note that a ground atom p picks out a set of subsets of X as its model class, namely the set of all supersets in X of the sense of p. Thus the model class of p is a member of the Boolean algebra determined by the power set of the power set of X with respect to union, intersection, and complement in 2^{2^X}. In order to complete the set up of the semantics for a spatially augmented language, we adopt a three-valued logic with truth values $\{\mathbf{t}, \mathbf{f}, \perp\}$. (Every sentence, i.e. the set of all formulas without free occurrences of variables, will turn out to have have a truth-value other than $\perp$ if every ground intensional atom has this property.) We adopt a standard set of strong interpretations of the 3-valued connectives [Kl67], pp. 334, derived from the standard 2-valued connectives of classical propositional logic where $\perp$ plays the role of *unknown*. It suffices to give the interpretation of $|$ i.e. *NAND*, or *not both*: $\mathbf{t} \mid \mathbf{t} = \mathbf{f}$, $\mathbf{f} \mid x = x \mid \mathbf{f} = \mathbf{t}$. The remaining pairs of inputs yield $\perp$. The interpretations of all other propositional combinations of truth values can be obtained by expressing the combination in terms of NAND as in the two-valued case. It is readily seen that the NAND expression one selects to represent a particular propositional connective is immaterial.

We inductively extend α_I to all of the elements of **Sent**, the set of sentences in a usual Tarskian manner. The existential quantifier is evaluated by the function α_I by: $\alpha_I(\exists x \varphi(x)) = \begin{cases} \mathbf{t} & \text{if } \alpha_I(\varphi(e/x)) = \mathbf{t} \text{ for some constant } e \\ \mathbf{f} & \text{if } \alpha_I(\varphi(e/x)) = \mathbf{f} \text{ for all constants } e \\ \perp & \text{otherwise.} \end{cases}$

The universal quantifier is treated as an abbreviation of $\neg\exists x \neg\varphi(x))$. Finally, we declare $I \models \varphi$ if and only if $\alpha_I(\varphi) = \mathbf{t}$.

A model, not necessarily stable, of a spatial program is a model of the set of all formulas in the EDB and IDB. Thus, in particular, a model of a program must contain the sense of every ground instance of each intensional atom in the EDB.

We note that if $\cap$ is the intensional connective corresponding to the intersection operator on 2^X and $A \cap B$ is a ground atom, then for $I \subseteq X$, there is no Boolean combination of the assertions $I \models A$ and $I \models B$ that holds if, and only if, $I \models A \cap B$ for all choices of the senses of A and B. Contrast this observation with: $I \models A \cup B$ if and only if $I \models A$ and $I \models B$.

3 The consequence operator and stable models

The following operator generalizes the one-step consequence-operator of ordinary logic programs with respect to 2-valued logic to spatial logic programs.
Given a spatial program $\mathcal{P}$ with IDB P, let P' be the set of ground instances of a clauses in P and let

$$T_P(I) = I_1 \cup I_2$$

where

$$I_1 = \bigcup \{\sigma(A) \mid A \leftarrow L_1, \ldots, L_n \ \in P', I \models L_i, i = 1, \ldots, n\} \text{ and}$$
$$I_2 = \bigcup \{\sigma(A) \mid A \text{ is a ground atom in the EDB of } \mathcal{P}.$$

A *supported model* of $\mathcal{P}$ a model of $\mathcal{P}$ that is a fixpoint of $T_{\mathcal{P}}$.

A spatial logic program is *Horn* if the IDB is Horn. Our definitions generalize the familiar characterization of the least model of ordinary Horn programs. However, if the Herbrand universe of a spatial program is infinite (contains infinitely many constants) then, unlike the situation with ordinary Horn programs, T_P will not in general be upward continuous.

We iterate T_P in the usual manner: $T_P \uparrow^0 (I) = I$,
$T_P \uparrow^{\alpha+1} (I) = T_P(T_P \uparrow \alpha(I))$, and
$T_P \uparrow^{\lambda} (I) = \bigcup_{\alpha<\lambda} \{T_P \uparrow \alpha(I)\}$, λ limit [1].

Example 1. To specify a spatial program we must specify the language, EDB and IDB. Let $\mathcal{L} = (L, X, \sigma, \mathcal{I})$ where L has four unary predicate symbols: p, q, r and s, and countably many constants $e_0, e_1, \ldots,$. X is the set $\mathbf{N} \bigcup \{\mathbf{N}\}$ where $\mathbf{N}$ is the set of natural numbers, $\{0, 1, 2, \ldots\}$. σ is specified by $\sigma(q(e_n)) = \{0, \ldots, n\}$, $\sigma(p(e_n)) = \{0, \ldots, n+1\}$, $\sigma(r(e_n)) = \mathbf{N}$, $\sigma(s(e_n)) = \{\mathbf{N}\}$.

The EDB is empty and the IDB is: $q(e_0) \leftarrow$, $p(X) \leftarrow q(X)$, and $s(e_0) \leftarrow r(e_0)$.

Now, after ω iterations upward from the empty interpretation, $r(e_0)$ becomes satisfied. One more iteration is required to reach an interpretation that satisfies $s(e_0)$, where the least fixpoint is attained.

It is clear that T_P is monotonic if P is a Horn program and thus that the following result follows from the Tarski fixpoint theorem.

Theorem 1. The least model of spatial Horn program P exists, is supported, and is given by $\mathbf{T}_P \uparrow^{\alpha} (\emptyset)$ for the least ordinal α at which a fixpoint is obtained.

What is different about the ascending iteration of T_P from the ordinary situation in logic programming is that in the spatial case the senses of ground body

[1] Batarakh and Subrahmanian, [BS89], studied applications of logic programming in lattices different from the lattice of interpretations.

atoms can be satisfied by the union of the senses of infinitely many ground clause heads without any finite collection of these clause heads uniting to satisfy the body atom. But, if there are only finitely many primitive atoms, i.e. the Herbrand *universe* of the program is finite, then this source of upward discontinuity vanishes. The proof of upward continuity is essentially the same in that case as the case for ordinary Horn programs.

Theorem 2. *The least model of spatial Horn program P exists, is supported, and is given by $\mathbf{T}_P \uparrow^{\omega} (\emptyset)$, if the set of primitive ground atoms in the Herbrand base of P is finite.*

In spatial logic programs, we allow clauses whose ground instances are of the following form:

$$A \leftarrow B_1, \ldots, B_n, \neg C_1, \ldots, \neg C_m. \tag{1}$$

We can the define the stable model semantics for such programs as follows. For any given set $J \subseteq X$, we define Gelfond-Lifschitz transform [GL88] of a program P, $GL(P)$, in two steps. First we consider all ground instances C of clauses in P as in (1). If $J \models C_i$ for some C_i in the body of C, then the we eliminate clause C. If not, then we replace C by the Horn clause

$$A \leftarrow B_1, \ldots, B_n. \tag{2}$$

The $GL(P)$ consists of $EDB(P)$ plus the sets of all Horn clauses produced by this two step process. Thus $GL(P)$ is a Horn program so that $T_{GL(P)}$ is defined. Then we say that J is *stable model* of P if and only if J equals the least model of $GL(P)$.

Theorem 3. *For any spatial logic program P,*

1. *$I \subseteq X$ is a model of $\mathcal{P}$ iff $T_P(I) \subseteq I$ and*
2. *I is stable with respect to $\mathcal{P}$ implies that I is supported with respect to $\mathcal{P}$.*

The next theorem shows the relationship between stable models of a a spatial program, and a natural topology induced by a spatial language on its interpretation space.

Theorem 4. *If $\mathcal{L}$ is a spatially augmented first-order language with the intensional operator for intersection of senses, then the set of senses of the ground intensional atoms form the basis of a topology in which all supported models, a fortiori all stable models, of all spatial programs over $\mathcal{L}$ are open subsets of the interpretation space.*

We will call the topology given by the previous theorem the *Herbrand topology.* This topology has a utility in finding stable models. Ordinarily one expects to recover a guess for a stable model as the least fixpoint of the Gelfond-Lifschitz transform determined by the guess. The previous theorem allows one to recover merely the interior of the guess, or equivalently, confine ones guesses to images of open sets. In the next section, where we incorporate miops into the one-step consequence operator of a program, we can achieve even greater selectivity of stable models.

4 Set Based Logic Programming with Miops

In this section, we shall introduce miops on the underlying space X of logic programming and show how we can extend the spatial logic programming paradigm of the previous section to incorporate miops.

Let X be the underlying space of spatial logic program P. We say that an operator $op : 2^X \to 2^X$ is a *miop* if for all $A, B \subseteq X$,

1. $A \subseteq B \implies op(A) \subseteq op(B)$ and
2. $op(op(A)) = op(A)$.

4.1 Operators and stable models

Suppose that the underlying space X is either $\mathbf{R}^n$ or $\mathbf{Q}^n$ were $\mathbf{R}$ is the reals and $\mathbf{Q}$ is the rationals. Then X is a topological vectors space under the usual topology so that we have a number of natural miop operators:

1. $op_{id}(A) = A$, i.e. the identity map is simplest miop operator,
2. $op_c(A) = \overline{A}$ where $\overline{A}$ is the smallest closed set containing A,
3. $op_{int}(A) = int(A)$ where $int(A)$ is the interior of A,
4. $op_{convex}(A) = K(A)$ where $K(A)$ is the convex closure of A, i.e. the smallest set $K \subseteq X$ such that $A \subseteq K$ and whenever $x_1, \ldots, x_n \in K$ and $\alpha_1, \ldots, \alpha_n$ are elements of the underlying field R or Q such that $\sum_{i=1}^n \alpha_i = 1$, then $\sum_{i=1}^n \alpha_i x_i$ is in K, and
5. $op_{subsp}(A) = (A)^*$ where $(A)^*$ is the subspace of X generated by A.

Now if we are given a miop operator $op^+ : 2^X \to 2^X$ and spatial logic program P over X, then we can further generalize the one-step consequence-operator of ordinary logic programs with respect to 2-valued logic to spatial logic programs relative to miop operator op^+ as follows. Given a spatial program $\mathcal{P}$ with IDB P, let P' be the set of ground instances of a clauses in P and let

$$T_{P,op^+}(I) = op^+(I_1 \cup I_2)$$

where
$I_1 = \bigcup\{\sigma(A) \mid A \leftarrow L_1, \ldots, L_n \ \in P', I \models L_i, i = 1, \ldots, n\}$.
$I_2 = \bigcup\{\sigma(A) \mid A \text{ is a ground atom in the EDB of } \mathcal{P}\}$.
A *supported model relative to* op^+ of $\mathcal{P}$ a model of $\mathcal{P}$ that is a fixpoint of T_{P,op^+}.

We iterate T_{P,op^+} according to the following.

$$\begin{aligned}
T_{P,op^+} \uparrow 0(I) &= I \\
T_{P,op^+} \uparrow \alpha + 1(I) &= T_{P,op^+}(T_{P,op^+} \uparrow \alpha(I)) \\
T_{P,op^+} \uparrow \lambda(I) &= op^+(\bigcup_{\alpha<\lambda} \{T_{P,op^+} \uparrow \alpha(I)\}),\ \lambda \text{ limit}
\end{aligned}$$

A spatial logic program is *Horn* if the IDB is Horn. Again it is easy to see that if P is a Horn program and op^+ is a miop, then T_{P,op^+} is monotonic. Thus just like in the case a spatial logic programs.

Theorem 5. Given a miop op^+, the least model of spatial Horn program P which is closed under op^+ exists, is supported relative op^+, and is given by $\mathbf{T}_{P,op^+} \uparrow^\alpha (\emptyset)$ for the least ordinal α at which a fixpoint is obtained.

Next we consider how we should deal with negation in the setting of miop operators. Suppose that we have a miop operator op^- on the underlying space X. In the definition of section 1, we say that $J \models \neg C_i$ if and only if it is not the case that $J \models C_i$. That is, $J \models \neg C_i$ if and only if $C_i \not\subseteq J$. As we mentioned in the introduction, it seem equally plausible to say that $J \models \neg C_i$ if and only if $J \cap C_i = \emptyset$. Thus we will define two different satisfaction relations for literal based relative to miop operator op^-[2]. This leads us to the following definition.

Definition 4. Suppose that $\mathcal{P}$ is spatial logic program over X and op^- is a miop operator on X.

(I) Given any atom C and set $J \subseteq X$, then we say $J \models^I_{op^-} \neg C$ if and only if $op^-(C) \cap J = op^-(\emptyset)$.
(II) Given any atom C and set $J \subseteq X$, then we say $J \models^{II}_{op^-} \neg C$ if and only if $op^-(C) \not\subseteq J$.

We can the define the two types of stable model semantics for a spatial logic program P over X relative to two miop operators op^+ and op^- on X. Let P be a spatial logic program over X and op^+ and op^- on X be two miop operators on X[3].

Definition 5. (**I**) For any given set $J \subseteq X$, we define Gelfond-Lifschitz transform of type I of a program P, $GL^I_{J,op^+,op^-}(P)$, in two steps. First we consider all ground instances of classes C in P as in (1). If it is not the case that $J \models^I_{op^-} \neg C_i$ for some i, then the we eliminate clause C. Otherwise we replace C by the Horn clause

$$A \leftarrow B_1, \ldots, B_n. \tag{3}$$

The $GL^I_{J,op^+,op^-}(P)$ consists of $EDB(P)$ plus the sets of all Horn clauses produced by this two step process. Thus $GL^I_{J,op^+,op^-}(P)$ is always a Horn program and hence $T_{GL^I_{J,op^+,op^-}(P),op^+}$ is defined. Then we say that J is **type I stable model** of P relative to (op^+, op^-) if and only if J equals the least model relative to op^+ of $GL^I_{J,op^+,op^-}(P)$.

(II) For any given set $J \subseteq X$, we define Gelfond-Lifschitz transform of type II of a program P, $GL^{II}_{J,op^+,op^-}(P)$, in two steps. First we consider all ground instances of classes C in P as in (1). If it is not the case that $J \models^{II}_{op^-} \neg C_i$ for some i, then the we eliminate clause C. Otherwise we replace C by the Horn clause

$$A \leftarrow B_1, \ldots, B_n. \tag{4}$$

[2] Lifschitz [Li94] observed that different modalities, thus different operators, can be used to evaluate positive and negative part of bodies of clauses of normal programs.
[3] It will often be the case that we take $op^+ = op^-$, but it is not required.

The $GL^{II}_{J,op^+,op^-}(P)$ consists of $EDB(P)$ plus the set of all Horn clauses produced by this two step process. Thus $GL^{II}_{J,op^+,op^-}(P)$ is always a Horn program and hence $T_{GL^{II}_{J,op^+,op^-}(P),op^+}$ is defined. Then we say that J is **type II stable model** of P relative to (op^+, op^-) if and only if J equals the least model relative to op^+ of $GL^{II}_{J,op^+,op^-}(P)$.

We then have the following result.

Theorem 6. *Suppose that $\mathcal{P}$ is spatial logic program over X and op^+ and op^- are miop operators on X. Assume I be closed relative to op^+, i.e., $op^+(I) = I$. Then*

1. *$I \subseteq X$ is a model of $\mathcal{P}$ iff $T_{\mathcal{P},op^+}(I) \subseteq I$.*
2. *I is stable of type I or II with respect to $\mathcal{P}$ implies that I is supported with respect to $\mathcal{P}$ relative to op^+.*

In the following subsections, 4.2-4.5, we shall give four examples to show how the stable models of a give spatial logic program can vary depending on how we define op^+ and op^-. We note that in the case where $op^- = op_{id}$ and the sense of any atom A such that $\neg A$ appears in $\mathcal{P}$ is a singleton, then there is no difference between the type I and type II stable models. Our examples in the next three subsections will all have this property so that we will not distinguish between type I and type II stable models.

4.2 Separating sets

Suppose that $V = Q^n$. Let $\mathbf{0}$ denote the zero vector of V. Suppose A and B are subsets of V. Our idea is construct a program whose stable models correspond to separating sets S such that S is closed relative to op^+, $A \subseteq S$ and $S \cap B = op^-(\emptyset)$. As we shall see that by picking the miop operators op^+ and op^- appropriately we can have a single spatial logic program P whose stable models have a variety of properties.

Formally, we shall assume that the underlying first order language has constant symbols a for each $a \in V$ and it has three unary predicate symbols S, $\overline{S}$ and A. Thus the ground atoms of the underlying Herbrand Base are all of the form $S(a)$, $\overline{S}(a)$ and $A(a)$ for some $a \in V$. We shall think of the interpretation space X as the set

$$X = \{S(a) : a \in V\} \cup \{\overline{S}(a) : a \in V\} \cup \{A(a) : a \in V\}.$$

The sense of any ground atom $S(a)$, $\overline{S}(a)$ and $A(a)$ will be just $\{S(a)\}$, $\{\overline{S}(a)\}$ and $\{A(a)\}$ respectively. That is: $\sigma(S(a)) = \{S(a)\}, \sigma(\overline{S}(a)) = \{\overline{S}(a)\}$ and$\sigma(A(a)) = \{A(a)\}$.

Now suppose that we are given a triple of miop operators $op_S, op_{\overline{S}}, op_A$ on V. Then we can define a miop operator op^+ on X as by defining op^+ so that

$$
\begin{aligned}
op^{+}(T) = \{S(a) : a \in op_S(\{y \in V : S(y) \in T\})\} \cup \\
\{\overline{S}(a) : a \in op_{\overline{S}}(\{y \in V : \overline{S}(y) \in T\})\} \cup \\
\{A(a) : a \in op_A(\{y \in V : A(y) \in T\})\}. \quad (5)
\end{aligned}
$$

The intuition here is that suppose we want in a stable model S and $\overline{S}$ to specify subspaces of V. Then we take $op_S = op_{\overline{S}} = op_{subsp}$ so that in any stable model the sets $\{a \in V : S(a) \in M\}$ and $\{a \in V : \overline{S}(a) \in M\}$ are subspaces. Now consider the following program $\mathcal{P}$.

(1) $S(a) \leftarrow$ for all $a \in A$.
(2) $\overline{S}(b) \leftarrow$ for all $b \in B$.
(3) $A(\mathbf{0}) \leftarrow S(x), \overline{S}(x), \neg A(\mathbf{0})$
(4) $S(x) \leftarrow \neg \overline{S}(x)$
(5) $\overline{S}(x) \leftarrow \neg S(x)$

We note that when we ground $\mathcal{P}$, the clauses of type (3), (4) and (5) will generate the following sets of ground clauses.

(3)' $A(\mathbf{0}) \leftarrow S(v), \overline{S}(v), \neg A(\mathbf{0})$ for all $v \in V$
(4)' $S(v) \leftarrow \neg \overline{S}(v)$ for all $v \in V$
(5)' $\overline{S}(v) \leftarrow \neg S(v)$ for all $v \in V$

Before proceeding we should make an observation about the clauses of type (3)' under the assumption that $op_A = op^- = op_{id}$. That is, if $op^- = op_{id}$, then $op^-(\sigma(A(\mathbf{0}))) = \{A(\mathbf{0})\}$. Now if $\{A(\mathbf{0})\} \subseteq M$, then it is not the case that $M \models^{I}_{op^-} \neg A(\mathbf{0})$ nor is it the case that $M \models^{II}_{op^-} \neg A(\mathbf{0})$. Note that every clause of $ground(\mathcal{P})$ which has $A(\mathbf{0})$ in the head has $\neg A(\mathbf{0})$ in the body. We claim that no matter how we define op_S, $op_{\overline{S}}$ and op^-, it will be that case that any type I or type II stable model M of $\mathcal{P}$ will have $\{a : A(a) \in M\} = \emptyset$. That is, since the only clauses which have an $A(v)$ in the head come from the clauses of type (3)', it automatically follows that it must be the case that $\{a : A(a) \in M\}$ is either equal to $op_A(\emptyset) = \emptyset$ or $op_A(\{A(\mathbf{0})\}) = \{A(\mathbf{0})\}$. But it cannot be that $A(\mathbf{0}) \in M$ since otherwise all the clauses of type (3)' will be eliminated when we take $GL^{II}_{op^+,op^-}(P)$ or $GL^{II}_{op^+,op^-}(P)$ and hence there would be no way to generate $A(\mathbf{0})$ by iterating $T_{GL^{I}_{op^+,op^-},op^+}$ or $T_{GL^{II}_{op^+,op^-},op^+}(P)$ starting at the empty set. Thus it must be the $\{a : A(a) \in M\} = \emptyset$. But then the effect of the clauses of type (3)' is to say that it is impossible that both $S(v)$ and $\overline{S}(v)$ are elements of a stable model M of $\mathcal{P}$ of type I or type II. Thus the effect of the clauses of type (3)' is to say that in any stable model M of type I or type II, when $op_A = op^- = op_{id}$, the sets $\{a : S(a) \in M\}$ and $\{a : \overline{S}(a) \in M\}$ are disjoint.

Next it is easy to see that the clauses of type (4)' and (5)' ensure that that for any stable model of type I or type II of $\mathcal{P}$, it is the case that $\{a : S(a) \in M\} \cup \{a : \overline{S}(a) \in M\} = V$. Similarly the clauses of type (1) and type (2) ensure that $A \subseteq \{a : S(a) \in M\}$ and $B \subseteq \{a : \overline{S}(a) \in M\}$. Thus it follows that no matter how we define op_S and $op_{\overline{S}}$, the set $\{a : S(a) \in M\}$ and $\{a : \overline{S}(a) \in M\}$, where M is stable model of type I or type II of $\mathcal{P}$, are a pair of separating sets for A and B.

Now consider the various options for op_S and $op_{\overline{S}}$. In each case stable models of $\mathcal{P}$ will characterize some desired class of sets. Moreover, in each case the stable models will be of the form $M = \{S(v) : v \in C\} \cup \{\overline{S}(v) : v \in X - C\}$ where C is the set which is characterized.

Proposition 1. *1. When $op_S = op_{id}$ and $op_{\overline{S}} = op_{id}$, $\mathcal{P}$ has a stable model if and only if $A \cap B = \emptyset$ and stable models of $\mathcal{P}$ characterize sets $C \subseteq V$ such that $A \subseteq C$ and $B \subseteq V - C$.*

2. *When $op_S = op_c$ and $op_{\overline{S}} = op_{int}$, $\mathcal{P}$ has a stable model if and only if $op_c(A) \cap B = \emptyset$ and stable models of $\mathcal{P}$ characterize closed sets $C \subseteq V$ and $A \subseteq C$ and $B \subseteq V - C$.*
3. *When $op_S = op_{int}$ and $op_{\overline{S}} = op_c$, $\mathcal{P}$ has a stable model if and only if $A \cap op_c(B) = \emptyset$. Stable models of $\mathcal{P}$ characterize open sets $C \subseteq V$ such that $A \subseteq C$ and $B \subseteq V - C$.*
4. *When $op_S = op_{conv}$ and $op_{\overline{S}} = op_{conv}$, $\mathcal{P}$ has a stable model if and only if $K(A) \cap K(B) = \emptyset$ and the stable models of $\mathcal{P}$ relative to op^+ characterize sets $C \subseteq V$ such that C and $V - C$ are convex, $A \subseteq C$ and $B \subseteq V - C$[4].*
5. *When $op_S = op_{subsp}$ and $op_{\overline{S}} = op_{id}$, $\mathcal{P}$ has a stable model if $op_{subsp}(A) \cap B = \emptyset$ and the stable models of $\mathcal{P}$ characterize subspaces of $C \subseteq V$ such that $A \subseteq C$ and $B \subseteq V - C$.*

4.3 Complementary subspaces

In this section we modify the previous construction to compute complementary subspaces. That is, suppose that we add 2 more predicates $T, \overline{T}$ and define $\sigma(T(v)) = \{T(v)\}$ and $\sigma(\overline{T}(v) = \{\overline{T}(v)\}$ for all $v \in X$. Next consider the program $\mathcal{Q}$ which is $\mathcal{P}$ from the previous construction plus the following set of clauses:

(6) $T(b) \leftarrow$ for all $b \in B$,
(7) $\overline{T}(a) \leftarrow$ for all $a \in A$,
(8) $A(\mathbf{0}) \leftarrow T(x), \overline{T}(x), \neg A(\mathbf{0})$,
(9) $T(x) \leftarrow \neg\overline{T}(x)$ and
(10) $\overline{T}(x) \leftarrow \neg T(x)$.

Then as before with shall take $op_A = op^- = op_{id}$, $op_S = op_T = op_{subsp}$ and $op_{\overline{S}} = op_{\overline{T}} = op_{id}$. Then we are essentially in the case of part 5 of the Proposition 1 so that all stable model of $\mathcal{Q}$ relative to op^+ are of the form

$$M = \{S(v) : v \in C\} \cup \{\overline{S}(v) : v \in X - C\} \cup \{T(v) : v \in D\} \cup \{\overline{T}(v) : v \in X - D\}.$$

[4] Recall the classical Convex Separation Theorem of Stone: if A and B are disjoint convex subsets of V, then there is a set C such that C and $V - C$ are convex subsets of V such that $A \subseteq C$ and $B \subseteq V - C$.

where C and D are subspaces of V such that $A \subseteq C$, $B \subseteq V - C$, $A \subseteq X - D$, and $B \subseteq D$. Finally we would like to add some clauses to ensure that C and D are complementary subspaces, i.e. $C \cap D = \{\mathbf{0}\}$ and $op_{subsp}(C \cup D) = V$. We add three more predicates U, $\overline{U}$ and equality where $op_U = op_{subsp}$ and $op_{=} = op_{\overline{U}} = op_{id}$. Consider the following clauses (11)-(18):

(11) $A(\mathbf{0}) \leftarrow U(x), \overline{U}(x), \neg A(\mathbf{0})$,
(12) $U(x) \leftarrow \neg\overline{U}(x)$,
(13) $\overline{U}(x) \leftarrow \neg U(x)$,
(14) $A(\{\mathbf{0}\}) \leftarrow \overline{U}(x), \neg A(\{\mathbf{0}\})$,
(15) $U(x) \leftarrow S(x)$,
(16) $U(x) \leftarrow T(x)$,
(17) $=(v,v) \leftarrow$ for all $v \in V$ and
(18) $A(\mathbf{0}) \leftarrow S(x), T(x), \neg(x = \mathbf{0}), \neg A(\mathbf{0})$.

By our previous analysis, clauses (11), (12) and (13) ensure that in a stable model M the set $E = \{v \in V : U(v) \in M\}$ is a subspace and $V - E = \{v \in V : \overline{U}(v) \in M\}$. Clause (14) then ensures that in a stable model $V - E = \{v \in V : \overline{U}(v) \in M\} = \emptyset$ and hence it must be the case that $E = \{v \in V : U(v) \in M\} = V$. However the only way that we can generate in E is via applications the clauses of the form (15) and (16) so that in a stable model, we must have $op_{subsp}(C \cup D) = E = V$. Finally the clauses of the form (17) and (18) ensure that $C \cap D = \{\mathbf{0}\}$. Thus we have the following result.

Proposition 2. *The stable models of program Q determine sets C and D to be complementary sub*spaces *of V.*

4.4 Continuous real functions

In this section, we shall use set based programming to write a program whose stable models represent continuous functions $F : [0,1] \rightarrow [0,1]$.

Let $\mathcal{R}$ be the real line, equipped with its usual topology. Let $\mathcal{R}^+$ be the set of all positive real numbers. Let ω be the set of all natural numbers. Let $\mathbf{Z}^+$ be the set of all positive integers.

It is easy to see that there $\mathcal{R}$ has a countable base $\{\, U_a \mid a \in \omega \,\}$ such that

1. $U_0 = \mathcal{R}$,
2. for each $a > 0$, U_a is an open interval (p_a, q_a) whose endpoints are dyadic rational,
3. the endpoint sequences $\langle p_a \rangle_{a \in \omega}$ and $\langle q_a \rangle_{q \in \omega}$ are computable,
4. there is a monotone function $e : \mathbf{Z}^+ \longrightarrow \mathbf{Z}^+$ such that, for each positive integer m, for each $a > e(m)$, the diameter of U_a is less than 2^{-m} and
5. for all natural numbers a and b, if $U_a \subset U_b$, then $a \geq b$.

For any positive integer n, the product space $\mathcal{R}^n$ also possesses such a base (with the obvious difference that the sets U_n for $n > 0$ are products of open intervals and that there are $2n$ computable sequences of endpoints.) Obviously, such a construction can be relativized to the product spaces $[0,1]^n$

Given such a base for the topology of $\mathcal{R}^n$, we can represent a continuous function $F : \mathcal{R}^n \longrightarrow \mathcal{R}^n$ by the function $f : \omega \longrightarrow \omega$ defined by

$$f(a) = \text{the greatest } b \text{ such that } F(U_a) \subset U_b \tag{6}$$

We can recover F from f, since, for each $x \in \mathcal{R}^n$, $F(x)$ is the unique member of $\bigcap_{a\in\omega, x\in U_a} U_{f(a)}$

Conversely, given such a base $\{\, U_a \,|\, a \in \omega \,\}$ and an arbitrary function $f : \omega \longrightarrow \omega$, it is natural to ask when is it the case that there is a continuous function $F : \mathcal{R}^n \to \mathcal{R}^n$ such that F is defined from f via (6). One can show that it is the case that for any fixed $x \in \mathcal{R}^n$,

$$\bigcap_{a\in\omega, x\in U_a} U_{f(a)}$$

is a singleton if and only if there is a function $d_x : \mathbf{Z}^+ \longrightarrow \mathcal{R}^+$ such that, for each natural number m and each positive integer k, $x \in U_m$ and U_m has diameter less than d_x implies $U_{f(m)}$ has diameter less than 2^{-k}. Thus, (6) defines a function from $\mathcal{R}^n$ to $\mathcal{R}^n$ if and only if such a d_x exists for every $x \in \mathcal{R}^n$. In the case of compact space like $[0,1]$, it is the case the (6) defines a function from $[0,1]$ to $[0,1]$ if and only if such a d, called the modulus of continuity of the F function, such that for all k, U_m has diameter less than $d(k)$ implies $U_{f(m)}$ has diameter less than 2^{-k}.

We shall consider a simplified version of this type of phenomenon. For example, let

$$\mathcal{A}_n = \{A_{n,k} : k = 0, \ldots, 2^n - 1\} \cup \{B_{n,k} : k = 1, \ldots, 2^{n-1} - 2\}. \tag{7}$$

where

$$A_{n,k} = \begin{cases} [0, \frac{1}{2^n}) & \text{if } k = 0, \\ (\frac{2^n-1}{2^n}, 1] & \text{if } k = 2^n - 1 \text{ and} \\ (\frac{k}{2^n}, \frac{k+1}{2^n}) & \text{if } k = 1, \ldots, 2^{n-1} - 2 \end{cases}$$

and

$$B_{n,k} = (\frac{2k+1}{2^{n+1}}, \frac{2k+3}{2^{n+1}}) \text{ for } k = 0, \ldots, 2^{n-1} - 1.$$

The significance of the family $\mathcal{A}_n$ is that every $x \in [0,1]$ lies in an open interval I of diameter $1/2^n$ for some $I \in \mathcal{A}_n$. Now suppose that our representing function f of a continuous function $F : [0,1] \to [0,1]$ has the property that if $U_a \in \mathcal{A}_{2n}$, then $f(a) = b$ where $b \in \mathcal{A}_n$. Thus the modulus of continuity function in this case is given by $d(k) = \frac{1}{2^{2k+2}}$. That is, whenever U_m has diameter $< d(k)$, $U_m \subseteq U_t$ where $U_t \in \mathcal{A}_{2k}$ and hence $U_{f(m)} \subseteq U_{f(t)} \in \mathcal{A}_k$ and hence $diam(U_{f(m)}) \leq diam(U_{f(t)}) = 2^{-k}$. In fact, it easy to see that can specify F by merely defining f on the a such that $U_a \in \bigcup_{n\geq 1} \mathcal{A}_{2n}$.

This given, we consider the following program. The constants of the underlying program will be $A_{n,k}$ such that $k = 0, \ldots, 2^n - 1$ and $n \geq 1$ and $B_{n,k}$ such that $k = 0, \ldots 2^{n-1} - 2$ for $n \geq 1$. Our program will contain one binary relation symbol $f(x, y)$ and one 0-ary predicate symbol C. The sense of a ground atom $f(E_{m,k}, F_{n,l})$ where $E, F \in \{A, B\}$ will simply be the open set $E_{m,k} \times F_{n,l}$ contained in $[0,1] \times [0,1]$. The sense of C is just $\{C\}$ so that the underlying space X consists of all $\{C\} \cup \{U_a \times U_b : a, b \in \omega\}$. We will take the miop operator $op_f = op_C = op^- = op_{id}$. Then consider the following propositional set-based program $\mathcal{P}$.

(1) $C \leftarrow f(X, Y), \neg C$ for all $X \in \mathcal{A}_{2n}$ and $Y \notin \mathcal{A}_n$,
(2) $C \leftarrow f(X, Y), f(X, Z), \neg C$ for all $X \in \mathcal{A}_{2n}$ and $Y, Z \in \mathcal{A}_n$ with $n \geq 1$ and $X \neq Y$,
(3) $C \leftarrow f(X, U), f(Y, V), \neg C$ for all $X, Y \in \bigcup_{n \geq 1} \mathcal{A}_{2n}$ and $U, V \in \bigcup_{n \geq 1} \mathcal{A}_n$ such that $X \subseteq Y$ but $U \not\subseteq V$.
(4) $f(X, Y) \leftarrow \neg f(X, U_1), \ldots, \neg f(X, U_{2^n + 2^{n-1} - 1})$ for each $n \geq 1$, $X \in \mathcal{A}_{2n}$ and $Y \in \mathcal{A}_n$ where $\mathcal{A}_n - \{Y\} = \{U_1, \ldots, U_{2^n + 2^{n-1} - 1}\}$ and
(5) $C \leftarrow \neg f(X, U_0), \ldots, \neg f(X, U_{2^n + 2^{n-1} - 1}), \neg C$ for each $n \geq 1$, $X \in \mathcal{A}_{2n}$ and $Y \in \mathcal{A}_n$ where $\mathcal{A}_n = \{U_0, \ldots, U_{2^n + 2^{n-1} - 1}\}$.

It is then easy to see by the same type analysis that we used in example 1, that C can never be an element of a stable model M for $\mathcal{P}$. It follows that effect of the clauses in (1), (2), (3) is to ensure that we can think of f as specifying a function defined on some subset of $\bigcup_{n \geq 1} \mathcal{A}_{2n}$ such that for each $n \geq 1$, (i) $X \in \mathcal{A}_{2n}$ implies $f(X) \in \mathcal{A}_n$ and (ii) if $X \subseteq Y$, then $f(X) \subseteq f(Y)$. Finally the clauses of (4) and (5) say that f must be defined on all of $\bigcup_{n \geq 1} \mathcal{A}_{2n}$. Thus we have the following

Proposition 3. *The stable models of* $\mathcal{P}$ *correspond to* $f : \bigcup_{n \geq 1} A_n \to \bigcup_{n \geq 1} A_n$ *such that for all* n, $a \in A_{2n}$ *implies* $f(a) \in A_n$ *and hence all such* f *define a continuous functions* $F : [0,1] \longrightarrow [0,1]$ *via (6).*

We should note that we did not really need to used set-based programming in this case as we could do the same thing in normal logic programming. The reason for presenting this construction is that by setting it in this framework, we can reason directly about the approximating interval $U_a \times U_{f(a)}$ to the function F in this case. Moreover, the framework of representing functions allows to reason about continuous transformations between different agents. Of course, in actual practice, we can only reason about approximations of continuous functions since continuous functions and/or their representing functions are infinite objects. In our setting, we can reason about approximations of representing functions by fixing some n_0 and restricting our program clauses so that all indices involved must be greater than or equal to n_0.

4.5 Distinguishing type I and type II stable models

We end this section with an example where there is a difference between type I and type II stable models. Suppose that the underlying space $X = \mathcal{R}^2$ is the real plane. Our program will have two atoms $\{a, b\}, \{c, d\}$ where a, b, c and d are reals. We let $[a, b]$ and $[c, d]$ denote the line segments connecting a to b and c to d respectively. We let sense of the these atoms be the corresponding subsets, i.e. we let $\sigma(\{a, b\}) = \{a, b\}$ and $\sigma(\{c, d\}) = \{c, d\}$. We let $op^+ = op^- = op_{convex}$. The consider the following program $\mathcal{P}$.

(1) $\{a, b\} \leftarrow \neg\{c, d\}$
(2) $\{c, d\} \leftarrow \neg\{a, b\}$

There are four possible candidate for stable models in this case, namely (i) $\emptyset$, (ii) $[a, b]$, (iii) $[c, d]$, and (iv) $op_{convex}\{a, b, c, d\}$.

If we are considering type I stable models where $J \models^I_{op^-} \neg C$ if and only if $op^-(C) \cap J = op^-(\emptyset) = \emptyset$, then the only case where there are stable models if $[a, b]$ and $[c, d]$ are disjoint in which (ii) case and (iii) are stable models.

If we are considering type II stable models where $J \models^{II}_{op^-} \neg C$ if and only if $op^-(C) \subsetneq J$, then there are no stable models if $[a, b] = [c, d]$, (ii) is stable model if $[a, b] \subsetneq [c, d]$, (iii) is stable model if $[c, d] \subsetneq [a, b]$ and (ii) and (iii) are stable models if neither $[a, b] \subseteq [c, d]$ nor $[c, d] \subseteq [a, b]$.

5 An application: cooperating multi-agents

In this section we will illustrate the power of spatial programs and miops to represent multi-agent systems by building upon an example given by Russell and Norvig [RN03] involving doubles tennis. Our point of departure is their 2-player doubles tennis team and how the team attempts to handle the return of an incoming ball. We first describe their representation of this situation.

Russell and Norvig set up two agents (the players on the team) that can each be in one of four discrete position values: *[Left, Baseline], [Right, Baseline], [Left, Net], [Right, Net].* Initially the ball is approaching the *[Right, Baseline]* position and it is assumed that the ball can only be returned from the position it is approaching. The goal is to return the ball and have both players positioned at the net.[5] Each player has three distinct actions available that can be invoked under certain preconditions. The effect of an action is to change the environment. In this tennis example, an environment is an association of values to the ball's approaching position attribute, the ball's *returned* attribute, and each agent's position attribute. The actions available to an agent are *NoOp* which has no effect, *Go* which reassigns the agent's position attribute value, and *Hit* which sets the ball's returned attribute to *true*. Since we are about to change the preconditions for moving a player anyway, we omit the description of the rather common sense preconditions for each of the actions.

[5] Each player can go to any position that she herself does not currently occupy, hence the position goal for the players is actually superfluous.

We will now modify Russell and Norvig's example to allow for the continuous motion of the players in continuous time and allow them a continuum of positions on their side of the court, as well as seek to prevent collisions when hitting the ball. We will represent the modified situation with a spatial logic program and selected miops. The representation will be accomplished by first considering a representation of the tennis example in a simplified variation of a fragment of William Rounds's and Hosung Song's ϕ-calculus [RS03], and then we will discuss how to represent the ϕ-calculus description as a spatial program with miops. We emphasize that we are not attempting to give a general procedure for representing ϕ-calculus models as spatial logic programs with miops. Rather, we are informally indicating that there is a relationship between ϕ-calculus (and similar calculi such as CCS and π-calculus) and spatial programs with miops.

We give our ϕ-calculus variation grammatically and then describe an operational semantics.

We have *Concurrent Expressions (CE)*, *Copy Expressions (YE)*, *Committed Choice Expressions (CCE)*, *Action Sequence Expressions (ASE)*, *Action Expressions (AE)*, *Environmental Actions (EA)*, and *messages (M)*. The grammar relating these types of phrases is given by the following production rules: $CE \longrightarrow CCE$, $CE \longrightarrow YE$, $CE \longrightarrow CCE \| CE$, $YE \longrightarrow\ !CCE$, $CCE \longrightarrow ASE$, $CCE \longrightarrow ASE + CCE$, $ASE \longrightarrow AE$, $ASE \longrightarrow AE\,.\,CCE$ (not quite hierarchical with this rule), $AE \longrightarrow EA$, $AE \longrightarrow M$, and each *message* is of the form *id* or $\overline{id}$ [6] where *id* is an identifier. An empty *CE* is denoted by *0*, a process which is unable to carry on any more actions. An EA is either empty, or has the form $[\varphi \to \mathcal{E}]$, where φ is a sentence from a fixed given spatially augmented first order language and $\mathcal{E}$ is an environment (to be discussed momentarily). *ASE* expressions are read, and processed, from left to right. The processing of *CE* expressions is made by considering all the subexpressions simultaneously; and the processing of *CCE* expressions consists on selecting only one of its subexpressions for execution. The *stopping condition* of a CE is the disjunction of the stopping conditions of its components. Analogously, the stopping condition of a CCE is the disjunction of the stopping conditions of the *AE*'s in the CE. The stopping condition of an EA, $[\varphi \to \mathcal{E}]$, is φ, and is called the *precondition* of the EA. The stopping condition of a message is *true*. The stopping condition of 0 is *true*. Finally, the stopping condition a YE is the stopping condition of its top-level CCE.

An *environment* is a quadruple (V, I, L, P) where V is a a k-tuple (for some k) of variables, I is a *set* of k-tuples of values in the interpretation space of the given spatially augmented language $\mathcal{L}$ for the variables in V, L is a *flow law* and P is an *invariant condition*, which must remain true while the state I evolves in accordance with the flow laws. Formally, I is a relation (which evolves as the environment evolves) on an interpretation space, the flow law L is a function that maps pairs consisting of, for some k, a k-ary relation on an interpretation space and a time (i.e. a real number) to k-ary relations on the interpretation

[6] In the literature, *id* is regarded as the action of listening for the message **id**, and $\overline{id}$ as the action of sending that message

space, and P is a formula of the given spatially augmented language $\mathcal{L}$ whose free variables are in V.

Consider the Euclidean plane with a standard coordinate system and the system of ordinary differential equations $\dot{x} = f(x, y)$, $\dot{y} = g(x, y)$. Take for I a small region such as the disc of radius 1 around the origin. The flow law is the system of ordinary differential equations. A suitable flow law can be thought of as causing the points in I to move to new positions as a function of time, thus moving and distorting I as a function of time. (It is possible to give e.g. differential equations that move at least some of the points of I to infinity after a short time, thus rendering the image of I at all later times undefined. We will assume that the admissible flow laws define an image of any subset of the plane at all times.) Think of the image of I evolving over time while the image satisfies or remains in the region specified by P. For example, the unit disc may move and distort for a while until the image no longer contains the origin. When that happens and is detected, we could stop the evolution of the image of I. In general, we can stop the evolution of the image of I at the greatest lower bound of all times t, at or after the starting time, at which P is satisfied by the image of I. Call this time the *stop time.*

Now, the essential connection with spatial logic programs and miops is that the image of I at the stop time, as a function of I, for a given admissible flow law and a given invariant condition, is a miop provided we ensure that any invariant conditions we use always eventually must be falsified such as by arranging for default stop times to be specified. If I increases in size, so do the images of I over time (monotonicity) and if a stop time has been reached, a restart with the same invariant condition goes nowhere (idempotency). Any computations of miops, such as those involving numerical solutions of ordinary differential equations and detections of invariant condition satisfaction are back-end computations. Using invariant conditions that allow the back-end computation to not return allows a form of deadlock. It is, therefore, reasonable to avoid such invariant conditions.

A sentence of the given spatially augmented language. A miop μ can be derived from L and an invariant condition φ as follows: $\mu_{L,\varphi}(I) = L(I, t_1)$ where $t_1 = \inf\{t \mid L(I, t) \models \neg\varphi\}$.

An operational semantics for our variant fragment of ϕ-calculus is based on a simple notion of concurrent threads, each of which is a sequence of actions to be carried out on an environment. Each element of a concurrent expression (CE) is either a copy expression (YE) or a committed choice expression (CCE). We defer describing the processing of copy expressions until the tennis example. Copy expressions have the least priority in processing. Think of a CE as a collection of concurrent queues, each of which is contained in a CCE. A CCE is an exclusive-or of these queues. A CCE is eligible to be selected for processing if its stopping condition is satisfied by the current environment. An eligible CCE is selected nondeterministically, but subject to the constraint that $CCE's$ containing an action sequence that can participate in a handshake (see below) have priority. An action sequence expression whose stopping condition is currently satisfied is selected from the CCE to replace the CCE, i.e. the CCE is *reduced*, again subject to the constraint that handshakes get priority. The operation is vacuous if the selected CCE is already an ASE, and we must build into interpreters the avoidance of repeated vacuous selections of reduced $CCE's$. An ASE is thought of as a queue. When an ASE is selected for processing, the action at the front of it (which may involve a handshake with the action at the front of another ASE) is processed in the manner of the π-calculus. Actions other than handshakes involve syntactic changes to the environment and resetting of variables. If there are no eligible $CCE's$, the environment is allowed to evolve in accord with its current flow law and invariant. If the environment's invariant condition is not satisfied and there are no eligible CCE's, deadlock results.

An operational semantics treats the phrase types in accordance with the following: Let $(\mathcal{E}, E)$ be a pair, where $\mathcal{E}$ is an environment and E is a CE. The stopping condition φ of the pair is the disjunction of the negation of the invariant condition of $\mathcal{E}$ and the stopping condition of E. A transition from $(\mathcal{E}, E)$ to $(\mathcal{E}', E')$ nondeterministically occurs, from the highest to the lowest priority, if:

- the expression E is a *concurrent expression* of the form $E_1 \| \ldots \| E_n$, and at least two subexpressions E_i and E_j are ready to synchronize by means of complementary messages. This action requires the structure of the subexpressions to be $E_i = a.\hat{E}_i$ and $E_j = \overline{a}.\hat{E}_j$. E' then results from E in the manner of π-calculus transitions, and $\mathcal{E}' = \mathcal{E}$. This is known as a *handshake*.
- the stopping condition of $(\mathcal{E}, E)$ is satisfied by the set in $\mathcal{E}$, none of the components of E are ready for a handshake , $\mathcal{E}'$ is the environment of an environmental action whose precondition is satisfied by the set in $(\mathcal{E}, E)$ and E' is a replacement of E in the manner of the π-calculus.
- the stopping condition of $(\mathcal{E}, E)$ is not satisfied by the set in $\mathcal{E}$, $E' = E$, and the set in $\mathcal{E}'$ is the result of applying the miop derived from the flow law in $(\mathcal{E}, E)$ and the stopping condition of $(\mathcal{E}, E)$ to the set in $(\mathcal{E}, E)$.

Whenever the expression E can be processed in the manner of the π-calculus, no environmental evolution can take place and E is processed. Otherwise $\mathcal{E}$ satisfies no active stopping conditions and is permitted to evolve according to its constituent flow law. The distinction between *committed choice* and mere *choice* is that any of the message components or action components of a CE with a satisfied precondition can be selected to be processed (with matching messages having priority) and then eliminated. With a CCE one of the action components with a satisfied precondition can be selected for processing after which the entire CCE is eliminated. Copy expressions are treated by regarding a CE as a list of components and list-processing the CE by performing an in-place replacement of a component $!CCE$ by $CCE \| !CCE$.

The first two types of transitions between environment/expression pairs itemized above are representable as clauses. We can construct a set-based logic program to be an interpreter for our ϕ-calculus fragment variant in accordance with the following. We want stable models to be *plans* which we represent as lists of pairs, each pair of the form $(\mathcal{E}, E)$, and consecutive pairs $(\mathcal{E}, E)$, $(\mathcal{E}', E')$ occurring in a plan only if $(\mathcal{E}, E) \mapsto (\mathcal{E}', E')$ is a legitimate transition. Note that whenever $\mathcal{E} \neq \mathcal{E}'$, it must be that the set component of $\mathcal{E}'$ results by applying a miop representing a flow or an environmental action to the set component of $\mathcal{E}$. The specific miop to be applied is determined by the top few levels of subexpressions in E together with the set component (which may be a singleton) in $\mathcal{E}$. We may regard the miop as a function of both $\mathcal{E}$ and E; in that case there is only a single miop π to represent environmental evolution that we need to be concerned with. We can identify the set components of $\mathcal{E}$ and, similarly, $\mathcal{E}'$, with a tuple of constants in a manner similar to a our treatment of vector spaces in section 4.2, whenever it is a singleton, as it is in the applications under discussion here. The sense of an atom $\textsc{trans}(\mathcal{E}, E, \mathcal{E}', E')$ is just the singleton of the pair $((\mathcal{E}, E), (\mathcal{E}', E'))$. We then declare a miop op to act on sets of these pairs by $op(R) = \{(s,t) \in R \mid t \in \pi(\{s\})\}$, where $R \subseteq S \times T$.

The set-up of a spatial logic program-based interpreter involves the need for, given a nonempty relation q in a program P, arranging a relation p such that in each stable model p is true of exactly one tuple and q is true of that tuple. The arrangement should not change the extension of q in any stable model. Specifically, we can add clauses to P as follows:

That in any stable model p will have at most one true instance is expressed by:

$$A(X,Y) \leftarrow p(X), p(Y), \neg A(X,Y)$$
$$A(X,X)$$

That in any stable model p and $\overline{p}$ are disjoint is expressed by:

$$A \leftarrow p(X), \overline{p}(X), \neg A$$

p and $\overline{p}$ are complementary:

$$p(X) \leftarrow \neg\overline{p}(X)$$
$$\overline{p}(X) \leftarrow \neg p(X)$$

That in any stable model that if p is true of X, then q is true of X is expressed in conjunction with the above clauses, while avoiding adding any new clauses with q occurring in their head, by:

$$\overline{p}(X) \leftarrow \neg q(X)$$

In conjunction with the above clauses, $\exists X p(X)$:

$$A \leftarrow \neg B, \neg A$$
$$B \leftarrow \neg\overline{p}(X)$$

The eight clauses given above can regarded as a macro, which we write as:

$$\text{exactlyOne}[q](X)$$

It is clear that we may generalize the single variable X to a tuple of variables. Moreover, by singling out a subtuple of variables among a tuple of variables, we may obtain (Skolem) functions. The following is the obvious abbreviation, where $Y_1, \ldots, Y_m$ functionally depends on $X_1, \ldots, X_n$:

$$\text{exactlyOne}[q](X_1, \ldots, X_n; Y_1, \ldots, Y_m)$$

Thus, given $a_1, \ldots, a_n$, we obtain exactly one instance of

$$\text{exactlyOne}[q](a_1, \ldots, a_n; Y_1, \ldots, Y_m)$$

true in any stable model.

In order to obtain one successful plan per stable model, we use the exactlyOne macro. Suppose we have set up, using standard techniques, a 1-ary predicate PLANPART which in any stable model is true of all and only the lists of pairs (Env, Exp) such that for two successive pairs (Env, Exp), (Env', Exp') in any list of which PLANPART is true, $\text{TRANS}(Env, Exp, Env', Exp')$ is true. Furthermore, suppose we have easily available list-processing predicates, $\text{FIRST}(A, L)$ true of all and only lists L whose first member is A in all stable models, and $\text{LAST}(Z, L)$ true of all and only lists L whose last member is Z in all stable models. (That lists are finite is easy to arrange in what we take for the interpretation space.) Then we may define the 3-ary predicate PLAN by

$$\begin{array}{l}\text{PLAN}((Env, Exp), (Env', Exp'), Plan) \leftarrow \\ \quad \text{FIRST}((Env, Exp), Plan), \text{LAST}((Env', Exp'), Plan), \text{PLANPART}(Plan)\end{array}$$

The macro

$$\text{exactlyOne}[\text{PLAN}]((Env, Exp), (Env', Exp'); Plan)$$

picks out exactly one $Plan$ starting from (Env, Exp) and ending with (Env', Exp') in any stable model of the interpreter for which $\text{PLAN}((Env, Exp), (Env', Exp'), Plan)$ is true.

We conclude this section with a discussion tracing the processing of the expression representing the double tennis team's attempt to return an incoming ball.

The representation of the tennis situation in our variant of the ϕ-calculus is:

$[P_1$ can reach ball $\rightarrow$ set P_1 to intercept ball $].\overline{\text{go}P_1}$.
 $([P_1$ can hit$\rightarrow$ stop P_1, return ball$].\overline{\text{hitBall}P_1}$ + $[P_1$ missed $\rightarrow$ all stop$]$. $\overline{\text{missed}P_1})$
$\|$
goP_2.$[P_2$ closer than $P_1 \rightarrow$ no reset$]$.
 $([true \rightarrow$ stop $P_1]$ + $[true \rightarrow$ move P_1 to default position$])]$
$\|$
$[P_2$ can reach ball $\rightarrow$ set P_2 to intercept ball $]$. $\overline{\text{go}P_2}$.
 $([P_2$ can hit $\rightarrow$ stop P_2, return ball $]$. $\overline{\text{hitBall}P_2}$ + $[P_2$ missed $\rightarrow$ all stop $].\overline{\text{missed}P_2})$
$\|$
goP_1.$[P_1$ closer than $P_2 \rightarrow$ no reset $]$.
 $([true \rightarrow$ stop $P_2]$ + $[true \rightarrow$ move P_2 to default position$])$

In order to simplify the representation, we will use labels to refer to the environmental actions in this process; actions with labels e_i belong to expressions related to player 1, and labels of the form e'_j refer to *EA*s of player 2.

Let:

e_1= [P_1 can reach ball $\rightarrow$ set P_1 to intercept ball]
e_2=[P_1 can hit$\rightarrow$ stop P_1, return ball]
e_3=[P_1 missed $\rightarrow$ all stop]
e_4=[P_2 closer than P_1 $\rightarrow$ no reset]
e_5=[*true* $\rightarrow$ stop P_1]
e_6=[*true* $\rightarrow$ move P_1 to default position])]
e'_1=[P_2 can reach ball $\rightarrow$ set P_2 to intercept ball]
e'_2=[P_2 can hit $\rightarrow$ stop P_2, return ball]
e'_3=[P_2 missed $\rightarrow$ all stop]
e'_4=[P_1 closer than P_2 $\rightarrow$ no reset]
e'_5=[*true* $\rightarrow$ stop P_2]
e'_6=[*true* $\rightarrow$ move P_2 to default position]

The representation of the tennis game is now:

$$\begin{array}{l} e_1.\overline{\mathrm{go}P_1}.(e_2.\overline{\mathrm{hitBall}P_1} + e_3.\overline{\mathrm{missed}P_1}) \parallel \\ \mathrm{go}P_2.e_4.(e_5 + e_6) \parallel \\ e'_1.\overline{\mathrm{go}P_2}.(e'_2.\overline{\mathrm{hitBall}P_2} + e'_3.\overline{\mathrm{missed}P_2}) \parallel \\ \mathrm{go}P_1.e'_4.(e'_5 + e'_6) \end{array}$$

When the ball approaches the players, the first and third components of the representation can potentially have their preconditions satisfied. It is presumed that at most one of the players can reach the ball. If neither *reach*-precondition is satisfied as the environment evolves the ball's position, then no expression processing will result, and the environment will eventually stop evolving when its default stopping condition is reached , for instance, when the ball goes out of the court.

Alternatively, suppose the *reach* precondition for P_1 is satisfied first. Then the environment evolution is halted, variable values are reset, and the flow law is altered to cause player 1 to intercept the ball when activated. The expression e_1 = [P_1 can reach ball $\rightarrow$ set P_1 to intercept ball] is popped and discarded. The expression now has the form:

$$\begin{array}{l} \overline{\mathrm{go}P_1}.(e_2.\overline{\mathrm{hitBall}P_1} + e_3.\overline{\mathrm{missed}P_1}) \parallel \mathrm{go}P_2.e_4.(e_5 + e_6) \parallel \\ e'_1.\overline{\mathrm{go}P_2}.(e'_2.\overline{\mathrm{hitBall}P_2} + e'_3.\overline{\mathrm{missed}P_2}) \parallel \mathrm{go}P_1.e'_4.(e'_5 + e'_6) \end{array}$$

At this point in the expression processing, the messages $\overline{\mathrm{go}P_1}$ and $\mathrm{go}P_1$ match and are eliminated by means of a *handshake* action. As mentioned above, this kind of action takes precedence over any other possible processing currently available in the expression. The result of the *handshake* is the expression:

$$\begin{array}{l} (e_2.\overline{\mathrm{hitBall}P_1} + e_3.\overline{\mathrm{missed}P_1}) \parallel \mathrm{go}P_2.e_4.(e_5 + e_6) \parallel \\ e'_1.\overline{\mathrm{go}P_2}.(e'_2.\overline{\mathrm{hitBall}P_2} + e'_3.\overline{\mathrm{missed}P_2}) \parallel e'_4.(e'_5 + e'_6) \end{array}$$

No other *handshake* is possible at the moment, so the environment resumes evolving until (possibly after 0 elapsed time) one of the preconditions [P_1 can hit] (from e_2), [P_2 can reach ball] (from e'_1), or [P_1 closer than P_2] (from e'_4) is satisfied.

Suppose for example that [P_2 can reach ball], the precondition of e'_1, is satisfied first. The environment is updated by using the information provided in e'_1 setting player 2 to go also after the ball. e'_1 is then eliminated from the expression, which now presents this form:

$$(e_2.\overline{\text{hitBall}P_1} + e_3.\overline{\text{missed}P_1}) \parallel \text{go}P_2.e_4.(e_5 + e_6) \parallel \\ \overline{\text{go}P_2}.(e'_2.\overline{\text{hitBall}P_2} + e'_3.\overline{\text{missed}P_2}) \parallel e'_4.(e'_5 + e'_6)$$

Notice how the processing of e'_1 has enabled a *handshake*, with messages $\text{go}P_2$ and $\overline{\text{go}P_2}$, which has to be processed before the environment can resume evolution. The expression is now:

$$(e_2.\overline{\text{hitBall}P_1} + e_3.\overline{\text{missed}P_1}) \parallel e_4.(e_5 + e_6) \parallel \\ (e'_2.\overline{\text{hitBall}P_2} + e'_3.\overline{\text{missed}P_2}) \parallel e'_4.(e'_5 + e'_6)$$

Both players will be attempting to reach the ball. Assuming that the ball is not close enough so neither player can determine whether they can reach it –preconditions for e_2 and e'_2– (much less, hit the ball), nor the players are able to decide which one is closer to the ball –preconditions for e_4, and e'_4 – the environment is allowed to evolve thus describing the trajectory of the ball and that of the players moving towards it.

Suppose that next [P_1 closer than P_2], the precondition of e'_4, is satisfied. Then, environmental evolution stops and the processing of e'_4 opens up a choice regarding the future actions for P_2 by exposing the *CCE* expression $e'_5 + e'_6$. e'_4 does not carry any resetting information for updating the current environment. This action is designed to detect the occurrence on an event for which its response is delegated to a subexpression. After removing e'_4 the expression has the form:

$$(e_2.\overline{\text{hitBall}P_1} + e_3.\overline{\text{missed}P_1}) \parallel e_4.(e_5 + e_6) \parallel \\ (e'_2.\overline{\text{hitBall}P_2} + e'_3.\overline{\text{missed}P_2}) \parallel e'_5 + e'_6$$

The processing of e'_4 paused the evolution of the environment. However, this will not start evolution again until a choice is made between e'_5 or a'_6. This situation is due to the fact that the preconditions of both e'_5 and a'_6 are *true*. After a choice is made, P_2 will remain in its current position if e'_5 is processed; or it will be discretely relocated a to a default position after processing e'_6. In either case, since no more actions follow e'_5 and e'_6, this subexpression is regarded as a null process 0 and it can safely be removed from this expression. After choosing either *EA* and properly updated the current environment, the expression is then:

$$(e_2.\overline{\text{hitBall}P_1} + e_3.\overline{\text{missed}P_1}) \parallel e_4.(e_5 + e_6) \parallel \\ (e'_2.\overline{\text{hitBall}P_2} + e'_3.\overline{\text{missed}P_2})$$

Evolution of the environment then resumes as P_1 attempts to intercept and hit the ball. It is important to realize that even though the expression above gives hope for P_2 to hit the ball (by means of the expression $e'_2.\overline{\text{hitBall}P_2} + e'_3.\overline{\text{missed}P_2}$), the current environment makes this possibility very unlikely: the current flow law of P_2 prevents it from moving after choosing either e'_5 or e'_6 in the previous step. It can be also verified that the preconditon for e_4 (P_2 closer than P_1) is not satisfied as P_1 is the only player going towards the ball and it was already decided that P_2 was not getting any closer to the ball.

P_1's attempt to intercept the ball may result in a successful return if e_2 is process. The environment then will be updated to stop P_1 and to change the direction of the ball, result of an effective return by P_1. In this case the expression will have the form:

$$\begin{array}{l}\overline{\text{hitBall}P_1} \parallel e_4.(e_5 + e_6) \parallel \\ (e'_2.\overline{\text{hitBall}P_2} + e'_3.\overline{\text{missed}P_2})\end{array}$$

On the other hand, P_1 may not be able to reach the ball, in which case the precondition of e_3 is designed to detect such event and the result of the processing of this EA will reflect P_1's failure. The expression is then:

$$\begin{array}{l}\overline{\text{missed}P_1} \parallel e_4.(e_5 + e_6) \parallel \\ (e'_2.\overline{\text{hitBall}P_2} + e'_3.\overline{\text{missed}P_2})\end{array}$$

In either case, the resulting expressions reflect a state in which no more progress can be made: a *handshake* involving $\overline{\text{hitBall}P_1}$ (or $\overline{\text{missed}P_1}$, in the second case) is not possible; and, no precondition of any of the EA availables is satisfied. In other words, the expression is functionally equivalent to the 0 process. It is clear that the representation of a complete tennis game will require the ability of allowing the players to react to more than one serv, game and set. For instance, we can embed each of the components of the original concurrent expression CE inside copy expressions CY. This would allow the creations of copies of those components as needed. The original expression would have had the following structure:

$$\begin{array}{l}!(e_1.\overline{\text{go}P_1}.(e_2.\overline{\text{hitBall}P_1} + e_3.\overline{\text{missed}P_1})) \parallel \\ !(\text{go}P_2.e_4.(e_5 + e_6)) \parallel \\ !(e'_1.\overline{\text{go}P_2}.(e'_2.\overline{\text{hitBall}P_2} + e'_3.\overline{\text{missed}P_2})) \parallel \\ !(\text{go}P_1.e'_4.(e'_5 + e'_6))\end{array}$$

The first step of the processing of this expression consists in obtaining a copy of each components based on the ϕ-calculus congruence rule (which it is imported from π-calculus) $!P \equiv P \| P$. The resulting expression is then:

$$\begin{array}{l}e_1.\overline{\text{go}P_1}.(e_2.\overline{\text{hitBall}P_1} + e_3.\overline{\text{missed}P_1}) \parallel \\ \text{go}P_2.e_4.(e_5 + e_6) \parallel \\ e'_1.\overline{\text{go}P_2}.(e'_2.\overline{\text{hitBall}P_2} + e'_3.\overline{\text{missed}P_2}) \parallel \\ \text{go}P_1.e'_4.(e'_5 + e'_6) \parallel \\ !(e_1.\overline{\text{go}P_1}.(e_2.\overline{\text{hitBall}P_1} + e_3.\overline{\text{missed}P_1})) \parallel \\ !(\text{go}P_2.e_4.(e_5 + e_6)) \parallel \\ !(e'_1.\overline{\text{go}P_2}.(e'_2.\overline{\text{hitBall}P_2} + e'_3.\overline{\text{missed}P_2})) \parallel \\ !(\text{go}P_1.e'_4.(e'_5 + e'_6))\end{array}$$

The representation is now in a form that allows us to proceed as described before.

6 Conclusions and Further Work

In this paper, we defined a variant of logic programming, called spatial logic programming, where the atoms have an associated *sense* (which is a subset of a given space) and have illustrated how such program can be used to naturally express problems in the various continuous domains. We envision many other applications of our spatial logic programming formalism such areas as graphics, image compression, and other domains where there are natural representation of processes that accept subsets of spaces as inputs and compute outputs, subsets of those spaces. Spatial programs with miops provide a logic-based approach to hybrid dynamical systems.

This paper is the first of a series of papers that will explore the spatial logic programming paradigm. For example there are a number of concepts from logic programming such as, *well-founded model* [VGRS91], stratified programs, etc. that can be lifted to the present context almost verbatim. Thus one can develop a rich theory of spatial logic programs. Our spatial logic programs share certain features with Constraint Logic Programming [JM94] and the exact connections need to be explored. Third, one can think about the *senses* of atoms as annotations of the kind discussed in [KS92]. While there are various differences between our approach and [JM94], for instance our use of negation as means to enforce constraints as in [Nie99], the relationship between these two approaches should be explored. Fourth, spatial logic programming can be studied in the more general setting of nonmonotone inductive definitions [Den00] (e.g. iterated inductive definitions of Feferman [Fef70]).

Acknowledgments: The second author has been partly supported by NSF grants IIS-0097278 and IIS-0325063. The authors wish to thank David Jakel for contributions and valuable discussion, particularly in regard to the description of the representation of continuous functions on the real numbers.

References

[ABW88] Apt, K., Blair, H.A., and Walker, A. Towards a theory of Declarative Knowledge. In *Foundations of Deductive Databases and Logic Programming*, J. Minker, Ed. Morgan Kaufmann, 89–148, 1988.

[BMR01] Blair. H.A., Marek, V.W. and Remmel, J.B. "Spatial Logic Programming" in *Proc. SCI 2001*, Orlando, FL, July, 2001.

[BS89] A. Batarekh and V.S. Subrahmanian. Topological Model Set Deformations in Logic Programming, Fundamenta Informaticae, Vol. XII, No. 3, pps 357–400, Sep. 1989, North Holland.

[Den00] Denecker, M. Extending classical logic with inductive definitions. In *First International Conference on Computational Logic (CL2000).* Lecture Notes in Artificial Intelligence, vol. 1861. Springer, 703–717, 2000.

[Fef70] Feferman, S. Formal theories for transfinite iterations of generalized inductive definitions and some subsystems of analysis. In *Intuitionism and Proof theory*, A. Kino, J. Myhill, and R. Vesley, Eds. North Holland, 303–326, 1970.

[GL88] Gelfond, M. and Lifschitz, V. The stable model semantics for logic programming. In *Proc. of the International Joint Conference and Symposium on Logic Programming.* MIT Press, 1070–1080, 1988.

[Li94] V. Lifschitz. Minimal belief and negation as failure. *Artificial Intelligence* 70:53–72, 1994.

[JM94] J. Jaffar and M. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19-20:503–581, 1994.

[KS92] Kifer, M. and Subrahmanian, V.S. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming* 12:335–367, 1992.

[Kl67] Kleene, S.C. *Introduction to Metamathematics*, North-Holland, 1967.

[Nie99] Niemelä, I. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence 25,* 3,4, 241–273, 1999

[RN03] Russell, Stuart J. and Peter Norvig Artificial Intelligence: A Modern Approach (2nd Edition). Prentice Hall, 2003.

[RS03] Rounds, William, C. and Hosung Song The Φ-Calculus: A language for distributed control of reconfigurable embedded systems. In *Hybrid Systems: Computation and Control: 6th International Workshop, HSCC 2003*, O. Maler and A. Pnueli, Eds. Lecture Notes in Computer Science, Springer Verlag, 435–449, 2003.

[VGRS91] Van Gelder, A., Ross, K., and Schlipf, J. The Well-Founded Semantics for General Logic Programs. *Journal of the ACM 38,* 3, 620–650, 1991.