Spatial Logic Programming

Howard A. Blair[1], Victor Marek[2], Jeffrey Remmel[3]

Abstract

We show that the mechanism underlying Logic Programming can be extended to handle the situation when the atoms are interpreted as subsets of various spaces. In such situation, the atoms of the underlying language are interpreted as subsets of the corresponding space. The models of the program are interpreted as subsets of that space. It turns out that the operator approach to Logic Programming can be lifted to such situation, and that when the Horn fragment of such logic is considered, the corresponding operators are monotone and reach the fixpoint in at most ω steps. The concepts of supported and stable models of programs can equally be generalized. We show how a classic PERT/CPM optimization problem can be represented in such circumstances.

1. Introduction

The motivation for this paper is to develop a logic programming paradigm which is capable to directly reason about regions in space as might be required for applications in graphics or image compression or to reason about time intervals as might be required in various job scheduling tasks. Thus instead having the intended underlying universe be the Herbrand base of the program, we would like to have the underlying universe by some fixed space X and have the atoms of the program specify subsets of X, i.e. elements of the sets of all subsets of X which we denote by 2^X.

If we reflect for a moment on the basic aspects of logic programming with an Herbrand model interpretation [Ap90, Lloyd87], a slight change in our point of view shows that interpreting atoms as subsets of the Herbrand base is a natural thing to do. That is, 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 declaring that the *sense*, $\sigma(p)$, of a ground atom p is the set $\{p\}$ and $I \models p$ iff $\sigma(p) \subseteq I$. By this simple move, we have permitted ourselves to see the sense of an atom as a subset of the Herbrand base.

This given, 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 regard an X-model of p as any set $I \subseteq X$ such that $\sigma(p) \subseteq I$. As a basis for a model theoretic semantics this set-up 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, called *spatial logic programs*, to serve as front-ends to existing systems and still have a seamless model-theoretic semantics for the system as a whole.

In this paper we (1) present a formalism, *spatial logic programs* in accord with the above ideas, (2) present a a model-theoretic semantics of the formalism, (3) identify the stable-model semantics of spatial programs, and (4) and to illustrate the utility of spatial programs, apply them to a portion of the operations research problem of project scheduling in the context of PERT/CPM.

2. Spatial Logic Programs: syntax and semantics

The syntax of spatial logic programs is based on the syntax of the formulas of what we define as *spatially augmented first-order predicate 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 which 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 expres-

[1]EECS Department, Syracuse University, Syracuse, NY 13244: e-mail blair@ecs.syr.eduI

[2]CS Department, University of Kentucky, Lexington, KY 40506: e-mail marek@cs.uky.edu

[3]Mathematics Department, University of California at San Diego, La Jolla, CA 92903: e-mail jremmel@ucsd.edu

sive power of intensional connectives allows us to capture external functions and relations intrinsic to the domain of a spatial logic 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 extending the set of ground atoms and suitably extending the sense assignment.

Definition 2.1 *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 four 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, in any spatial logic program only finitely many of them will occur.

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 *extensional atoms* of $\mathcal{L}$ in the usual inductive manner. That is,
1) an atomic formula A of L, the underlying first-order language component of $\mathcal{L}$, is an extensional atom. The predicate symbol of A is the *principal functor* of A and
2) if $\varphi_1, \ldots, \varphi_n$ are extensional atoms and η is an n-ary intensional connective, then $\eta(\varphi_1, \ldots, \varphi_n)$ is an extensional atom, whose principal functor is η.

The remaining extensional formulas of $\mathcal{L}$ are built up from extensional atoms in the usual way. It should be noted that intersection is *not* representable as familiar Boolean connectives. This will become clear in the next section on semantics.

We can then extend the notion of sense to arbitrary intensional ground atoms inductively by declaring that an intensional ground atom A of the form $\eta(\varphi_1, \ldots, \varphi_n)$ and let the interpretation of η be a function $f : (2^X)^n \longrightarrow 2^X$. Then the sense of A, denoted by $\sigma(A)$, is recursively given by $f(\sigma(\varphi_1), \ldots, \sigma(\varphi_n))$.

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

Definition 2.2 *A* **spatial logic program** *has three components.*
1) The language $\mathcal{L}$ *which includes the interpretation space and the sense assignment.*
*2) The IDB (***Internal 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 extensional atom or the negation of an extensional atom, and* A *is an atomic formula of* L *whose principal functor does not occur as the principal functor of any ground extensional atom in the EDB.*
*3) The EDB (***External** *Database): A finite set of extensional ground atoms.*

For the rest of this paper, we shall assume that the classes of spatial logic programs that we consider always are over a language for first-order logic L without nonconstant function symbols, 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. HB_L is the Herbrand base of L, i.e. set of ground 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 $\sigma : HB \longrightarrow 2^X$. The subset of X, $\sigma(p)$, is called the **sense** of the ground atom p (with respect to X). An **interpretation** of the spatial language $\mathcal{L} = (L, X, \sigma, \mathcal{I})$ is a subset of X. A ground extensional atom p is satisfied by interpretation I, with respect to sense assignment σ (denoted by $I \models_\sigma p$) iff $\sigma(p) \subseteq I$.

The sense assignment σ can be given to partition the ground atoms into multiple sorts. For example, Let X be the disjoint union of X_1 and X_2. Let HB be the disjoint union of A_1 and A_2, and choose σ such that $\sigma(p) \subseteq X_i$ for $p \in A_i$, $i = 1, 2$. In particular, we may want to have some atoms reserved for having singleton senses.

The preceding definition allows us to extend the sat-

isfaction relation to all extensional 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 next extend the the satisfaction relation to arbitrary formulas. 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}.

1) If φ is a non ground formula, then $I \models \varphi$ iff $I \models \varphi'$ for every ground instance φ' of φ.

2) If φ is an extensional atom, then $I \models \varphi$ iff $\sigma(\varphi) \subseteq I$.

3) $I \models \varphi_1 \vee \varphi_2$ iff $I \models \varphi_1$ or $I \models \varphi_2$.

4) $I \models \varphi_1 \wedge \varphi_2$ iff $I \models \varphi_1$ and $I \models \varphi_2$.

5) $I \models \neg\varphi$ iff $I \not\models \varphi$.

6) $I \models \varphi_1 \leftarrow \varphi_2$ iff $I \models \varphi_1 \vee \neg\varphi_2$.

7) The remaining Boolean connectives are handled similarly.

8) $I \models \exists x\varphi(x)$ iff $I \models \varphi(e)$ for some constant symbol e of $\mathcal{L}$.

9) $I \models \forall x\varphi(x)$ iff $I \models \varphi(e)$ for all constant symbols e of $\mathcal{L}$.

We say that I **models** a set of of formulas Γ iff $I \models A$ for all $A \in \Gamma$.

An ordinary model 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 extensional atom in the EDB.

Note in particular 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: $I \models A \cup B$ iff $I \models A$ 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

$$T_P(I) = I_1 \cup I_2 \cup I_3$$

where

$I_1 = \bigcup\{\sigma(A) \mid A \leftarrow L_1, \ldots, L_n \in P', I \models L_i, i = 1, \ldots, n\}$.

$I_2 = \bigcup\{\sigma(A) \mid A \leftarrow \in P'$ and A is a ground external atom$\}$.

$I_3 = \bigcup\{\sigma(A) \mid A$ is a ground atom in the EDB of $\mathcal{P}\}$.

A *supported model* of $\mathcal{P}$ a model of $\mathcal{P}$ that is a fixed point of $\mathbf{T}_P$.

A spatial logic program is *Horn* if the EDB 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, $\mathbf{T}_P$ will not in general be upward continuous.

We iterate $\mathbf{T}_P$ according the following.

$$\begin{array}{lcl} \mathbf{T}_P \uparrow^0 (I) & = & I \\ \mathbf{T}_P \uparrow^{\alpha+1} (I) & = & \mathbf{T}_P(\mathbf{T}_P^{\alpha}(I)) \\ \mathbf{T}_P \uparrow^{\lambda} (I) & = & \bigcup\limits_{\alpha<\lambda} \mathbf{T}_P \uparrow \alpha(I)\},\ \lambda \text{ limit} \end{array}$$

Example 3.1 *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* $\sigma(q(e_n)) = \{0, \ldots, n\}$, $\sigma(p(e_n)) = \sigma(q(e_{n+1}))$, $\sigma(r(e_n)) = \mathbf{N}$, $\sigma(s(e_n)) = \{\mathbf{N}\}$.

The EDB is empty and the IDB is

$$\begin{array}{lcl} q(e_0) & \leftarrow & \\ p(X) & \leftarrow & q(X) \\ s(e_0) & \leftarrow & r(e_0) \end{array}$$

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 fixed point is attained. This completes the current example.*

It is clear that $\mathbf{T}_P$ is monotonic is clear so that the following result follows from Tarski's theorem on fix points of monotonic operators [Tarski55].

Theorem 3.1 The least model of spatial Horn program $\mathcal{P}$ exists, is supported, and is given by $\mathbf{T}_P \uparrow^{\alpha} (\emptyset)$ for some ordinal α.

What is different about the ascending iteration of $\mathbf{T}_P$ from the ordinary situation in logic programming is that in the spatial case, the senses of ground body atoms can be satisfied by the union of the senses of infinitely many ground clause heads without any finite collection of of these clause heads uniting to satisfy the body atom. But, if there are only finitely many non-external 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 then as the case for ordinary Horn programs.

Theorem 3.2 *The least model of spatial Horn program $\mathcal{P}$ exists, is supported, and is given by* $\mathbf{T}_P \uparrow^\alpha (\emptyset)$ *for the least ordinal α at which a fixed point is obtained.*

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

$$A \leftarrow B_1, \ldots, B_n, \neg C_1, \ldots, \neg C_m. \qquad (1)$$

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 of classes C in P as in (1). If $J \models C_i$ for some i, then the we eliminate clause C. If not, then we replace C by the Horn clause

$$A \leftarrow B_1, \ldots, B_n. \qquad (2)$$

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

Theorem 3.3 *1. $I \subseteq X$ is a model of $\mathcal{P}$ iff* $\mathbf{T}_P(I) \subseteq I$.

2. I is stable with respect to $\mathcal{P}$ implies that I is supported with respect to $\mathcal{P}$.

Proof: The proofs of the two parts of the above proposition proceed exactly as they do for ordinary programs after noting that for *any* $I \subseteq X$, an element x of $\mathbf{T}_P(I)$ is an element of the sense of at least one of the clause heads in $\mathcal{P}$ whose body is satisfied by I.

4. An application: PERT/CPM

In this section we illustrate spatial logic programming with a small but realistic application. We make no claims that spatial programming is the best approach to this application. Our purpose in presenting the example is to illustrate how spatial programs serve as a natural front-end to organize a a collection of perfectly well-understood pre-existing algorithms. Here *PERT* stands *P*roject/program *E*valuation and *R*eview *T*echnique and *CPM* stands for *C*ritical *P*ath *M*ethod) is a classical operations research planning activity [HL95].

The basic planning problem is: One is given a number of tasks, each assumed to have a minimum required execution time. One is also given an *immediate predecessor* relation among the tasks. The problem is to determine the earliest and latest starting and completion times relative to the overall starting time so as to complete the entire project in the minimum time.) In the standard presentation of the CPM technique there are three phases.

(1) In the first phase, one is to construct a *network diagram* to model the project. The network diagram is a directed acyclic graph. The network diagram's edges are the tasks of the project, together with certain other so-called *dummy* tasks that require no time to complete. The network diagram's nodes are thought of as moments when tasks are started and completed. These are called *events*. Since one can complete several tasks simultaneously in the model and then go on to start several other tasks that depended on these, the in-degree and out-degree of the events in the diagram have no a priori bounds. It is further assumed that for two events E_1 and E_2 there is at most one task that starts at E_1 and completes at E_2. The immediate predecessor relation is taken to mean that if task A immediately precedes B, then there is no task C that depended on A and upon which B depended. Furthermore, the transitive closure of the immediate successor relation (the converse of the given immediate predecessor relation) is assumed to be an irreflexive partial ordering. In order to maintain these requirements it is, in general, necessary to insert dummy tasks in to the network that require no time to complete.

(2) In the second phase there are two passes: one computes the earliest time to completion of the entire project by calculating the earliest time each event can occur. Then, working backwards from the earliest time to complete the project relative to the start time, one calculates the latest time at which any event can occur. An task in the project is *critical* if the latest time of occurrence and earliest time of occurrence coincide for both its starting and completing events, and the difference between these times is precisely the minimum time to complete the task. A *critical path* in the project is a path through the network diagram from the starting event for the entire project to the terminating event for the entire project along which every task is critical. A simple induction argument shows that at least one critical path always exists.

(3) In the third phase one identifies the critical paths and calculates the time intervals, i.e. float times, within which the various tasks may start while still completing the entire project in the minimum time.

The PERT technique differs from the CPM technique only in that it obtains a simple probabilistic estimate of critical paths and float times in contrast to the deterministic analysis of CPM. Our task is to craft a spatial program that, by varying the EDB component, we declaratively express each PERT/CPM project in a clear manner.

The first realization about PERT/CPM is that the first phase is altogether unnecessary. Activities actually undertaken are simply nonempty open intervals of time (assuming that the real tasks of the project require some nonzero minimum time to complete. It might also be that several tasks of the same duration occur simultaneously, but are not identical. We can model the situation with a multi-dimensional Euclidean space. Earliest and latest times of occurrence of events are just attributes of the tasks. Attributes of the tasks will be associated with their tasks by intensional connectives. The transitive closure of the converse of the immediate predecessor relation yields a directed acyclic graph in which the tasks occur as nodes, not edges, and the start and completion earliest and latest times can be computed just as straightforwardly as in the standard CPM presentation.

As we proceed we will identify the senses of ground atoms and identify the intensional connectives along with their interpretations. We will introduce the clauses of the IDB independently of any specific problem instance, and an EDB corresponding to a project with seven tasks.

The underlying first-order language: The language determined by the constants and predicate symbols occurring in the definitions, IDB and EDB. The predicate symbols, constants, and intensional connectives of the underlying first-order language will be introduced as we present the clauses. The interpretation space X is the disjoint union of the sets $\mathbf{N}$, Nat, $\mathbf{N}\times\mathbf{N}$, $\mathbf{N}\times\mathbf{R}$, $\mathbf{N}\times\mathbf{R}$, $\mathbf{N}\times\mathbf{R}$, $\mathbf{R}$, $\mathbf{R}$, and $\mathbf{R}\times\mathbf{R}$. We will use elements, as singletons, of $\mathbf{N}\times\mathbf{N}$ to be the senses of the ground atoms that represent dependencies between tasks. Senses of ground atoms representing the sets of stopping times of a set of tasks will be subsets of a $\mathbf{R}$ component. The senses of the atoms representing the starting and stopping times of particular tasks will be singleton subsets of the $\mathbf{N}\times\mathbf{R}$ components in the manner described in the sequel. Atoms representing the time intervals over which tasks are performed will have their senses be singleton subsets of of real numbers representing starting and stopping times. Certain instances of the atoms initial(X) and notInitial(X) will have their senses be singleton subsets of the respective $\mathbf{N}$ components.

Computation space: The set $\mathbf{R}^+$ defined by of nonnegative real numbers.

In our example we will represent the dependencies between seven given tasks and their durations with the following EDB:

$\mathsf{task}_0 \lhd \mathsf{task}_2$ $\quad$ $\mathsf{duration}(\mathsf{task}_0, \mathsf{t}_0)$
$\mathsf{task}_1 \lhd \mathsf{task}_2$ $\quad$ $\mathsf{duration}(\mathsf{task}_1, \mathsf{t}_1)$
$\mathsf{task}_1 \lhd \mathsf{task}_3$ $\quad$ $\mathsf{duration}(\mathsf{task}_2, \mathsf{t}_2)$
$\mathsf{task}_2 \lhd \mathsf{task}_5$ $\quad$ $\mathsf{duration}(\mathsf{task}_3, \mathsf{t}_3)$
$\mathsf{task}_3 \lhd \mathsf{task}_4$ $\quad$ $\mathsf{duration}(\mathsf{task}_4, \mathsf{t}_4)$
$\mathsf{task}_3 \lhd \mathsf{task}_6$ $\quad$ $\mathsf{duration}(\mathsf{task}_5, \mathsf{t}_5)$

$\mathsf{task}_4 \lhd \mathsf{task}_5$ $\quad$ $\mathsf{duration}(\mathsf{task}_6, \mathsf{t}_6)$

The sense of a ground atom of the form $\mathsf{task}_m \lhd \mathsf{task}_n$ is the pair of integers (m, n), and the sense of any other ground instance of $\mathsf{X} \lhd \mathsf{Y}$ is X.

The following clauses identify initial tasks.

$\mathsf{initial(T)} \leftarrow \neg\mathsf{notInitial(T)}$

$\mathsf{notInitial(T)} \leftarrow \mathsf{K} \lhd \mathsf{T}$

The sense of an atom of the form $\mathsf{initial}(\mathsf{task}_n)$ is $\{n\}$ in the first $\mathbf{N}$ component of X , and the sense of an atom of the form $\mathsf{notInitial}(\mathsf{task}_m)$ is is $\{m\}$ in the second $\mathbf{N}$ component. The sense of any other instance of either initial(X) or notInitial(X) is X.

The next clause says that the starting time of an initial job is time 0.

$start(J, 0) \leftarrow initial(J)$

The sense of $\mathsf{start}(\mathsf{task}_m,\mathsf{t}_m)$ and $\mathsf{stop}(\mathsf{task}_n,\mathsf{t}_n)$ are $\{(m, t_m)\}$ and $\{(n, t_n)\}$ in the two $\mathbf{N}\times\mathbf{R}$ components, respectively. X is the sense of any other instances of these predicates.

The next clause defines the stopping times of tasks in terms of starting times and durations.

$\mathsf{stop(J, A)} \leftarrow$

$\mathsf{start(J, B)}, \mathsf{duration(J, C)}, \mathsf{real(A)} + \mathsf{real(B)} = \mathsf{real(C)}$

The sense of an atom real(r) is the singleton $\{r\}$ for a constant r that we assign to represent the real number r, for each real number r. All other instances of the predicate real have X as their sense. The interpretation of the intensional connective + is the essentially the addition operation on the reals, but lifted to singleton sets of reals in the obvious way. The attempt to "add" X to a singleton set of reals results in X. The intensional connective interpretation denoted by $=$ returns the empty set if the two sets compared are equal, and otherwise returns X.

The following clauses recursively define the earliest starting time of a task T in terms of the maximum stopping time in the set of stopping times of tasks upon which T depends.

$\mathsf{enablingTimes(J, empty)} \leftarrow$

$\mathsf{enablingTimes}(J, \mathsf{singleton0}) \leftarrow \mathsf{initial}(J)$

$\mathsf{enablingTimes}(J, S) \leftarrow \mathsf{enablingTimes}(J, S'), K \lhd J, \mathsf{stop}(K, P),$

$\quad \mathsf{enablingTimes}(J, S') \cup \mathsf{real}(P) = \mathsf{enablingTimes}(J, S)$

$\mathsf{start}(J, T) \leftarrow \max \mathsf{enablingTimes}(J, S) = \mathsf{real}(T), \neg\mathsf{startOK}(J, S)$

$\mathsf{startOK}(J, T) \leftarrow \neg\mathsf{enablingTimes}(J, S'), T < \max(\mathsf{enablingTimes}(J, S'))$

The sense of ground atoms of the form $enablingTimes(task_n,s_n)$ is a subset $\{n\}\times S_n$ of $\mathbf{N}\times\mathbf{R}$, where S_n is a subset of $\mathbf{R}$, where s_n is a constant denoting a subset of $\mathbf{R}$, and the sense of any other instance of the enablingTimes predicate is X. The sense of ground instances of startOK is like that of ground instances of start, but using the remaining $\mathbf{N}\times\mathbf{R}$ component of X. It is at this point crucial to note that in any particular instance of a PERT/CPM problem, only finitely many of the constants denoting various real numbers and sets of reals will arise. Finally, the intensional connective $<$ returns the emptyset if the two sets compared are singleton sets of reals and the real number in the first argument is less than the real number in the second argument. Otherwise $<$ returns X.

5. Conclusions and Further Work

In this paper, we defined a variant of logic programming, called spatial logic programming, where the atoms are associated with *sense* (which is a subset of a given space) and have illustrated how such program can be used to naturally express problems in the PERT/CPM domain. 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.

This paper is the first of a series of papers that will explore the spatial logic programming paradigm. There are a number of areas and questions that could not be covered in this paper due to space considerations. 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 beteween our approach and [JM94], for instance our use of negation as means to enforce constraints as in [Niemelä99], the relationship between these two approaches should be explored. Fourth, spatial logic programming can be studied in the more general setting of non-monotone inductive definitions [Denecker00] (e.g. iterated inductive definitions of Feferman [Feferman70]).

References

[Ap90] Apt, K. R. "Logic Programming" in *Handbook of Theoretical Computer Science*, J. van Leeuwen, ed., Elsevier, pp. 494-574, 1990.

[ABW88] Apt, K., Blair, H., 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.

[Denecker00] 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.

[Feferman70] 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.

[HL95] Hillier, F.S. and Lieberman, G.L. *Introduction to Operations Research* 6th ed, McGraw-Hill, 1995.

[Lloyd87] Lloyd, J. *Foundations of Logic Programming.* Springer-Verlag, 1987.

[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.

[Niemelä99] 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

[Tarski55] Tarski, A. Lattice-theoretic fixpoint theorem and its applications. *Pacific journal of Mathematics 5*, 285–309, 1955

[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.