Proceedings of the International Conference on Knowledge Representation and Reasoning (KR), 2004.

A Logic of Motion

Fusun Yaman and **Dana Nau** and **V.S. Subrahmanian**
Department of Computer Science,
University of Maryland Institute for Advanced Computer Studies (UMIACS),
and Institute for Systems Research (ISR)
University of Maryland
College Park, MD, 20742
{ fusun,nau,vs}@cs.umd.edu

Abstract

There are numerous applications such as air traffic management, cellular phone location tracking, and vehicle protection systems where there is a critical need to reason about moving objects. In this paper, we propose a formal *logic of motion* (LOM for short). We provide a formal syntax for LOM, as well as a model theory for LOM. In addition, we develop algorithms to check consistency of LOM theories, as well as to answer certain kinds of queries posed to LOM theories. We have implemented these algorithms in a prototype LOM system - we describe experiments showing that such queries can be efficiently executed in practice.

Introduction

There are numerous applications in the world today that involve objects that are moving in space and time. Here are several examples:

- In the USA, the Federal Aviation Authority must reason, in real time, about the locations of various airplanes and helicopters in US airspace. Other countries have similar agencies as well.
- Wireless companies are continuously interested in where their customers are located and how they are moving - this information is used to estimate load on different cell phone towers and allocate handoff policies.
- A third application involves systems such as LOJACK and ONSTAR which are anti-theft devices used in vehicles. Here, monitoring stolen vehicles requires the ability to reason about where these vehicles are.
- We have built three applications for the US military which involve spatiotemporal reasoning about moving objects. The first application reasons about where and when enemy submarines will be in the future, while the second one is used to ensure that multiple US Navy ships will not collide in the future. The third one extends the US Army's *Combat Information Processor* that manages tactical battlefield simulations which tracks movements of friendly and enemy vehicles in a given region on the ground. Such a system associates an ID with each vehicle and estimates various parameters such as its current location, its current velocity, and so on.

Users may wish to pose many different kinds of queries to such systems. Such queries could include:

(Q1) What are all the vehicles within a given rectangular region R now?

(Q2) What are all the vehicles that might reach intersection I in the next 30 minutes?

(Q3) What are all the friendly vehicles that may be threatened by enemy vehicle e within the next 10 minutes (assuming the vehicles continue to move at the current speed and directions)? Here threatened might mean: be within 1 mile of e.

Though the dynamics of motion have been studied extensive since the time of Newton, to our knowledge, there is no *logic of motion* (LOM for short) which is used to formalize how vehicles move. This paper makes the following contributions:

1. It provides a formal syntax for *motion* theories and a fragment of motion theories called *go*-theories.
2. It provides a formal model theory for motion theories together with a result that checking consistency of *go*-theories is NP-complete.
3. In lieu of a classical Hilbert-style proof theory (which we defer to the journal version of this paper), we present sound and complete algorithms for checking consistency of *go*-theories, and for processing certain types of queries against *go*-theories. This is a *de facto* operational semantics for our logic of motion.
4. It gives experimental results showing the performance of LOM, a prototype system based on the above algorithms.

LOM Syntax

We assume the existence of two sets of constant symbols: $\mathbf{R}$ is the set of all real numbers and $\mathbf{OID}$ is the set of all object ids. We assume the existence of two disjoint sets $V_{\mathbf{R}}, V_{\mathbf{OID}}$ of variables ranging over $\mathbf{R}$ and $\mathbf{OID}$, respectively. We also assume the existence of three special predicate symbols go, near, in, of arities $11, 5, 7$ respectively. As usual a *real* (resp. object) term t (resp. o) is any member of $\mathbf{R} \cup V_{\mathbf{R}}$ (resp. $\mathbf{OID} \cup V_{\mathbf{OID}}$).

LOM Atoms. Let x_1, y_1, x_2, y_2, t_1^-, t_1^+, t_2^-, t_2^+, v^-, v^+, d, t_1, and t_2 be real terms; and let o, o_1, and o_2 be object terms.

Then the following are *LOM atoms*:

$$\mathsf{near}(o_1, o_2, d, t_1, t_2);$$
$$\mathsf{in}(o, x_1, y_1, x_2, y_2, \ell, h);$$
$$\mathsf{go}(o, x_1, y_1, x_2, y_2, t_1^-, t_1^+, t_2^-, t_2^+, v^-, v^+).$$

Ground terms and ground atoms are defined in the usual way.

Intuitively, $\mathsf{go}(o, x_1, y_1, x_2, y_2, t_1^-, t_1^+, t_2^-, t_2^+, v^-, v^+)$ is true if object o leaves location (x_1, y_1) at some time point in the interval $[t_1^-, t_1^+]$, goes ***along the straight line connecting*** (x_1, y_1) ***and*** (x_2, y_2) at a speed between v^- and v^+, and arrives at location (x_2, y_2) at some time point in the interval $[t_2^-, t_2^+]$. $\mathsf{in}(o, x_1, y_1, x_2, y_2, t_1, t_2)$ returns true if o is guaranteed to be inside the rectangle whose lower left corner is (x_1, y_1) and whose upper right corner is (x_2, y_2) (with vertical and horizontal edges) at some time point between t_1, t_2. Likewise, $\mathsf{near}(o_1, o_2, d, t_1, t_2)$ is true if objects o_1 and o_2 are within distance d of each other at all[1] times t, $t_1 \le t \le t_2$. $\mathsf{in}(o, x_1, y_1, x_2, y_2, \ell, h)$ is true if object o is inside the rectangle whose lower left corner is (x_1, y_1) and whose upper right corner is (x_2, y_2) at some time point at some time during the interval $[\ell, h]$. Two sides of the rectangle are assumed to be parallel to the x-axis and the other two to the y-axis here.

Note that unlike most existing studies of motion (whether in the AI or database or networking communities), this framework allows us to express uncertainty about when an object leaves a given location, when it arrives at its destination, and what its velocity is. This is consistent with the real world where the velocity of an object may vary (e.g. with traffic in the case of road vehicles, with windspeed in the case of aerial vehicles, with oceanographic currents in the case of marine vehicles, etc.). This in turn has an impact on exactly when a vehicle will reach its destination - that too is uncertain. We are not aware of existing treatments of this uncertainty in reasoning about moving objects.

LOM formulas are inductively defined as follows: every LOM atom is a LOM formula; and if F_1, F_2 are LOM formulas, then so are $F_1 \wedge F_2$, $F_1 \vee F_2$ and $\neg F_1$.

A *motion theory* is a finite set of LOM formulas. A *go-theory* is a finite set of *ground* go-atoms.[2]

Notation. If

$$g = \mathsf{go}(o, x_1, y_1, x_2, y_2, t_1^-, t_1^+, t_2^-, t_2^+, v^-, v^+),$$

[1]In (Yaman, Nau, & Subrahmanian 2004), we also describe a predicate symbol in which the "all" requirement here is replaced by "some."

[2]It is straightforward to generalize the definition of a motion theory to allow several other predicate symbols, higher order derivatives (e.g. acceleration), and motion in spaces having more than 2 dimensions. See (Yaman, Nau, & Subrahmanian 2004) for details.

then we will let

$$obj(g) = o;$$
$$loc_1(g) = (x_1, y_1);$$
$$loc_2(g) = (x_2, y_2);$$
$$t_1^-(g) = t_1^-;$$
$$t_2^-(g) = t_2^-;$$
$$t_1^+(g) = t_1^+;$$
$$t_2^+(g) = t_2^+,$$
$$v^-(g) = v^-;$$
$$v^+(g) = v^+.$$

LOM Semantics: Model Theory

In this section, we define a formal model theoretic semantics for LOM. A *LOM-interpretation* I is a continuous[3] function from $\mathbf{OID} \times \mathbf{R}$ to $\mathbf{R} \times \mathbf{R}$. Intuitively, $I(o, t)$ is the location of object o at time t. We will often abuse notation and write $I(o, [t_1, t_2])$ for a closed real interval $[t_1, t_2]$ to denote the straight line connecting the points $I(o, t_1)$ and $I(o, t_2)$.

Satisfaction of a ground LOM formula F by a LOM-interpretation I is defined as follows:

1. If $F = \mathsf{go}(o, x_1, y_1, x_2, y_2, t_1^-, t_1^+, t_2^-, t_2^+, v^-, v^+)$, then $I \models F$ iff the following conditions hold:
   - There are real numbers $t_1 \in [t_1^-, t_1^+]$ and $t_2 \in [t_2^-, t_2^+]$ such that $I(o, t_1) = (x_1, y_1)$, $I(o, t_2) = (x_2, y_2)$, and $I(o, t)$ maps the interval $[t_1, t_2]$ one-to-one onto the line segment $[(x_1, y_1), (x_2, y_2)]$. Intuitively, this means that during $[t_1, t_2]$, o moves from (x_1, y_1) to (x_2, y_2) without stopping, going in reverse, disappearing and reappearing, or "jumping" from one place to another in zero time.
   - At all but finitely many points in $[t_1, t_2]$, the derivative $v(t) = d(|I(o, t)|)/dt$ (which represents o's speed) is defined, and $v^- \le v(t) \le v^+$. However, we do not require I to be analytic (as in Newtonian physics), because in many practical applications it is useful to allow $I(o, t)$ to be, for example, piecewise linear.
2. If $F = \mathsf{near}(o_1, o_2, d, t_1, t_2)$, then $I \models F$ iff $\mathsf{dist}(I(o_1, t), I(o_2, t)) \le d$ for every $t_1 \le t \le t_2$, where dist is the function that computes the classical Euclidean distance between two points.
3. If $F = \mathsf{in}(o, x_1, y_1, x_2, y_2, \ell, h)$, then $I \models F$ iff there are numbers $t \in [\ell, h]$, $x \in [x_1, x_2]$, and $y \in [y_1, y_2]$ such that $I(o, t) = (x, y)$.
4. $I \models F \wedge G$ iff $I \models F$ and $I \models G$.
5. $I \models F \vee G$ iff $I \models F$ or $I \models G$.
6. $I \models \neg F$ iff I does not satisfy F.

I satisfies a motion theory MT iff I satisfies every $F \in$ MT. MT is *consistent* iff there is a LOM interpretation I such

[3]Continuity is w.r.t. classical real fields (Ellis & Gulick 1978) rather than, say, the discrete notion of continuity used in logic programming (Lloyd 1987).

that $I \models \mathsf{MT}$. F is a *logical consequence* of MT, denoted $\mathsf{MT} \models F$, iff every LOM interpretation I that satisfies MT also satisfies F.[4]

The following theorem describes the computational complexity of checking consistency.

Theorem 1 *The problem of checking whether an input motion theory is consistent is NP-hard. The problem is NP-complete if the input theory is required to be a* go-*theory.*

The proof of NP-hardness is obtained by polynomially reducing the problem of sequencing with deadlines and release times which is known to be NP-complete (Garey & Johnson 1977) to the problem of checking consistency of a motion theory.

In addition, checking consistency is complicated by the fact that a single $\mathsf{go}(obj_1, 0, 0, 0, 60, 44, 48, 50, 52, 4, 5)$ is inconsistent because there is no way to get from location $(0, 0)$ to location $(60, 44)$ within the prescribed speed limits of 4-5 miles per hour within the prescribed time frame (leave sometime in the $[44, 48]$ interval and arrive at some time in the $[50, 52]$ interval). In addition, consider the go-atom $\mathsf{go}(obj_1, 0, 0, 0, 60, 35, 50, 40, 55, 6, 10)$. The distance between the origin and destination is 60. If the object leaves at time 35, the earliest time at which it can arrive at the destination (given the max speed of 10) is at time 41. Thus the lower bound on the arrival time which is 40 in the above go-atom can be tightened to 41. Likewise, the upper bound may also be amenable to tightening.

Consistency Checking

In this section, we describe deterministic and nondeterministic algorithms to check consistency of *go*-theories.

Given a *go*-theory G and an object o, let G^o denote the set of all atoms $g \in G$ such that $obj(g) = o$. It is clear that G is consistent iff G^o is consistent for all objects o. *Therefore, throughout the rest of this paper, without loss of generality, we assume that for all $g_i, g_j \in G$, $obj(g_i) = obj(g_j)$. In other words, all atoms in G are about the same object.*

Coherent movement. Consider the following question: *Given a set of* go-*atoms G is it possible for G to describe the movement of a single object o on a single line segment over a single continuous time interval?* In this case, we say that the set G of atoms describes a coherent movement. Here are some examples, for the case where G consists of two go-atoms g_1, g_2:

1. Suppose $t_2^+(g_1) < t_1^-(g_2)$. Then the answer is no, because g_1 must end before g_2 starts.
2. Suppose $t_1^-(g_1) \le t_2^+(g_2)$, $t_1^-(g_2) \le t_2^+(g_1)$, the line segments $[loc_1(g_1), loc_2(g_1)]$ and $[loc_1(g_2), loc_2(g_2)]$ are not collinear or the line segments do not intersect or the directions of the movements[5] are not the same. Then the answer is no, because g_1 and g_2 must overlap temporally but they define incompatible trajectories for o during the time that they overlap.
3. Suppose $t_1^-(g_1) \le t_2^+(g_2)$, $t_1^-(g_2) \le t_2^+(g_1)$, the line segments $[loc_1(g_1), loc_2(g_1)]$ and $[loc_1(g_2), loc_2(g_2)]$ are collinear and their intersection is not empty and direction of the movements are same. Then the answer may be either yes or no, depending on whether the minimum and maximum speeds $v_1(g_1), v_2(g_1), v_1(g_2), v_2(g_2)$ are compatible or the common line segment can be visited at the same times by both g_1 and g_2. Since o's actual speed does not need to be constant, those conditions are rather complicated.

More generally, suppose $G = \{g_1, \ldots, g_n\}$ is a set of go-atoms object o (i.e., $\forall g_i \in G,\ obj(g_i) = o$) and such that the union of the line segments $\{[loc_1(g_i), loc_2(g_i)]\}_{i=1}^n$ is a single line segment $L = [P_1, P_2]$. Here are the formal conditions under which is it possible for G to consistently define the movement of o over a single time interval. Let $\{p_1, \ldots, p_n\} = \{loc_1(g_i), loc_2(g_i)\}_{i=1}^k$. Without loss of generality, assume that the points $p_1, \ldots, p_n$ are listed in ascending order of their distance from P_1. We use the notation $Movement(G)$ to denote the set of constraints given below.

1. $t_1^-(g_j) \le T_i \le t_1^+(g_j)$ for every i, j such that $p_i = loc_1(g_j)$;
2. $t_2^-(g_j) \le T_i \le t_2^+(g_j)$ for every i, j such that $p_i = loc_2(g_j)$;
3. $dist(p_i, p_{i+1}) \le (T_{i+1} - T_i) \times v_i^+,\ i = 1, \ldots, n-1$;
4. $(T_{i+1} - T_i) \times v_i^- \le dist(p_i, p_{i+1}),\ i = 1, \ldots, n-1$;

where

- $T_1, \ldots, T_n$ are variables;
- $v_i^- = \max\{v^-(g) \mid [p_i, p_{i+1}]$ is a subsegment of the line segment $[loc_1(g), loc_2(g)]\}$;
- $v_i^+ = \min\{v^+(g) \mid [p_i, p_{i+1}]$ is a subsegment of the line segment $[loc_1(g), loc_2(g)]\}$.

We will show later that consistency of a *go*-theory requires satisfying all the constraints in $Movement(G)$. The rationale behind this assertion is based on the intuition that T_i denotes the actual time at which the object o leaves point p_i. The first constraint above says that if g_j is any go-statement involving leaving from location p_i, then T_i must lie within the earliest departure time and the latest departure time from point p_i according to g_j. The second constraint says that if g is any go-atom that describes when o arrives at

[4]Suppose $G = \{g_1, g_2\}$, where

$$g_1 = \mathsf{go}(o, x_{11}, y_{11}, x_{12}, y_{12}, t_{11}^-, t_{11}^+, t_{12}^-, t_{12}^+, v_1^-, v_1^+);$$
$$g_2 = \mathsf{go}(o, x_{21}, y_{21}, x_{22}, y_{22}, t_{21}^-, t_{21}^+, t_{22}^-, t_{22}^+, v_2^-, v_2^+).$$

Suppose the points $(x_{11}, y_{11}), (x_{12}, y_{12}), (x_{21}, y_{21}), (x_{22}, y_{22})$ are all distinct and $t_{21}^- > t_{12}^+$. Then the above definition allows I to have the object o travel at an arbitrarily high speed during the open interval (t_{12}^+, t_{21}^-). We can prevent this by introducing a notion of V-interpretation where $V > 0$ is some real value that bounds the maximal velocity of object o. Replacing the notion of satisfaction and logical consequence by the notion of V-satisfaction and V logical consequence can be dealt with using similar methods to those described here. See (Yaman, Nau, & Subrahmanian 2004) for details.

[5]Direction of movement is a unit vector $(loc_2(g_1) - loc_1(g_1))/dist(loc_2(g_1), loc_1(g_1))$

p_i then T_i must lie within the times at which o can reach p_i as well. This is needed because otherwise we could not view the set of go-atoms in G as representing a continuous sequence of movements. The third and fourth constraints say that the arrival time of object o at p_{i+1} from point p_i must be compatible with the distance between these two points and the velocity of the object. Note that if multiple go-atoms cover the line segment between p_i and p_{i+1}, then the velocities in question must all apply to the movement of o from p_i to p_{i+1}.

Note that as $Movement(G)$ only contains linear constraints, there are Linear Programming (LP) solvers (Khachiyan 1979; Karmarkar 1984) that can solve $Movement(G)$ in polynomial time.[6]

Example 1 *If $g_1 = go(obj_1, 40, 10, 40, 60, 1, 5, 6, 14, 5, 10)$ and $g_2 = go(obj_1, 40, 30, 40, 90, 8, 10, 13, 18, 6, 12)$ then it is possible to combine them into a single movement, because $t_1^-(g_1) \leq t_2^+(g_2)$ and $t_1^-(g_2) \leq t_2^+(g_1)$, the line segments $[(40, 10)(40, 60)]$ and $[(40, 30)(40, 90)]$ are collinear with non-empty intersection $[(40, 30)(40, 60)]$ and direction of the movements are same. Furthermore, the following constraints (in $Movement(\{g_1, g_2\})$) are satisfiable:*

$$\begin{aligned}
1 \leq & T_1 \leq 5; \\
8 \leq & T_2 \leq 10; \\
6 \leq & T_3 \leq 14; \\
13 \leq & T_4 \leq 18; \\
dist(p_1, p_2) &\leq (T_2 - T_1) \times 10; \\
dist(p_2, p_3) &\leq (T_3 - T_2) \times 10; \\
dist(p_3, p_4) &\leq (T_4 - T_3) \times 12; \\
(T_2 - T_1) \times 5 &\leq dist(p_1, p_2); \\
(T_3 - T_2) \times 6 &\leq dist(p_2, p_3); \\
(T_4 - T_3) \times 6 &\leq dist(p_3, p_4);
\end{aligned}$$

where $p_1 = (40, 10)$, $p_2 = (40, 30)$, $p_3 = (40, 60)$ and $p_4 = (40, 90)$. A solution to the constraints above is $T_1 = 4$, $T_2 = 8$, $T_3 = 13$, $T_4 = 16$.

Consistency of a *go*-theory. We are now ready to consider the problem of consistency of an arbitrary *go*-theory. For simplicity, consider a pair of go-atoms g_1, g_2. Intuitively, there are three cases in which $\{g_1, g_2\}$ is consistent:

1. if it is possible to end g_1 before g_2 starts, which can happen iff $t_2^-(g_1) < t_1^+(g_2)$;
2. if it is possible to end g_2 before g_1 starts, which can happen iff $t_2^-(g_2) < t_1^+(g_1)$;
3. if it is possible for g_1 and g_2 to overlap, which can happen iff the line segments $[loc_1(g_1), loc_2(g_1)]$ and $[loc_1(g_2), loc_2(g_2)]$ are collinear, the line segments intersect, direction of the movements are same and $Movement(\{g_1, g_2\})$ has a solution.

[6]In (Yaman, Nau, & Subrahmanian 2004), we show that an even better time bound can be achieved. In polynomial time, $Movement(G)$ can be transformed into a Simple Temporal Problem (STP) (Dechter, Meiri, & Pearl 1991), and the satisfiability of the STP can be checked in $O(n^3)$. As discussed in (Dechter, Meiri, & Pearl 1991), minimum and maximum values for each T_i can be computed in $O(n^3)$ and a solution can be constructed within the same time complexity bounds.

We have generalized the above reasoning to get a nondeterministic polynomial-time algorithm Consistent(G) that determines whether a set of go-atoms $G = \{g_1, \ldots, g_n\}$ is consistent.

Algorithm Consistent(G)

1. Let C be a set of linear constraints that is initially empty. Let Γ be a graph whose nodes are $g_1, \ldots, g_n$, and whose edges are chosen nondeterministically from the set $\{\{g_i, g_j\} \mid$ the line segments $[loc_1(g_1), loc_2(g_1)]$ and $[loc_1(g_2), loc_2(g_2)]$ are collinear and intersect$\}$.
2. For every connected component s of Γ, insert $Movement(s)$ into C. For every pair of connected components s, s' of Γ, nondeterministically insert one of the following two sets of constraints into C:
   - $\{T < T' \mid T$ is a variable of s and T' is a variable of $s'\}$;
   - $\{T' < T \mid T$ is a variable of s and T' is a variable of $s'\}$.
3. If C has a solution then return "yes," else return "no."

To see that the algorithm runs in nondeterministic polynomial time, note that in every execution trace, Steps 1 and 2 end after a polynomial number of steps, and Step 3 can be done using a polynomial-time LP solver.

Just as with any nondeterministic polynomial-time algorithm, Consistent(G) can be translated into a sound and complete deterministic algorithm that runs in exponential time. We now present the deterministic version of Consistent(G), CheckConsistency(G).

```
Algorithm CheckConsistency(G)
Let $O = \{(g_i, g_j) \mid g_i$ and $g_j$ are go-atoms that
        can overlap$\}$
Let $\Gamma$ be a graph with no edges whose vertex set is $G$
Let $SP = \emptyset$
Let $C$ be an empty set of constraints
return SolveConstraints($O$, $\Gamma$,$SP$,$C$)
```

```
Algorithm SolveConstraints($O$, $\Gamma$,$SP$,$C$)
if $O \neq \emptyset$ then
    $(g_i, g_j)$=first element of $O$
    $O' = O - \{(g_i, g_j)\}$
    if SolveConstraints($O'$, $\Gamma$,$SP$,$C$) then
        return "yes"
    else
        $\Gamma' = \Gamma$ with additional edge $(g_i, g_j)$
        returnSolveConstraints($O$, $\Gamma'$,$SP$,$C$)
elseif $C = \emptyset$ then
    Let $S$ be the connected components of $\Gamma$
     Let $SP'$ be the set of all pairs in $S$
     Let $C' = \{movement(s) \mid s \in S\}$
     return SolveConstraints($O$, $\Gamma$,$SP'$,$C'$)
elseif $SP = \emptyset$ then
     return "yes" if $C$ is solvable, otherwise "no"
```

```
else
    $(s_1, s_2)$=first element of $SP$
    $SP' = SP - \{(s_1, s_2)\}$
    $C' = C+$ constraints of $s_1$ before $s_2$
    if SolveConstraints($O$, $\Gamma$,$SP'$,$C'$) then
        return "yes"
    else
        $C' = C+$ constraints of $s_1$ after $s_2$
        return SolveConstraints($O$, $\Gamma$,$SP'$,$C'$)
```

The CheckConsistency(G) algorithm calls SolveConstraints(O, Γ,SP,C) procedure, which performs a depth-first search over all possible alternatives. It is a recursive algorithm which in the first phase builds the graph Γ, then in the second phase selects an ordering over all connected components of Γ and finally checks to see if the constraints C is solvable. If at any point the algorithm returns "yes" all recursive calls return "yes" and the algorithm stops searching. On the other hand SolveConstraints(O, Γ,SP,C) returns "no" only when all choices return "no." The reader can verify that at every decision point there are only 2 choices and the depth of the recursion can be at most $O(n^2)$, thus the time complexity for CheckConsistency(G) is $O(2^{n^2})$.

Example 2 *Let*

$$g_1 = go(obj_1, 200, 300, 200, 500, 10, 20, 30, 70, 4, 10);$$
$$g_2 = go(obj_1, 200, 400, 200, 600, 20, 25, 40, 65, 5, 10);$$
$$g_3 = go(obj_1, 40, 10, 40, 60, 1, 5, 6, 14, 5, 10).$$

Then only g_1 and g_2 can overlap.

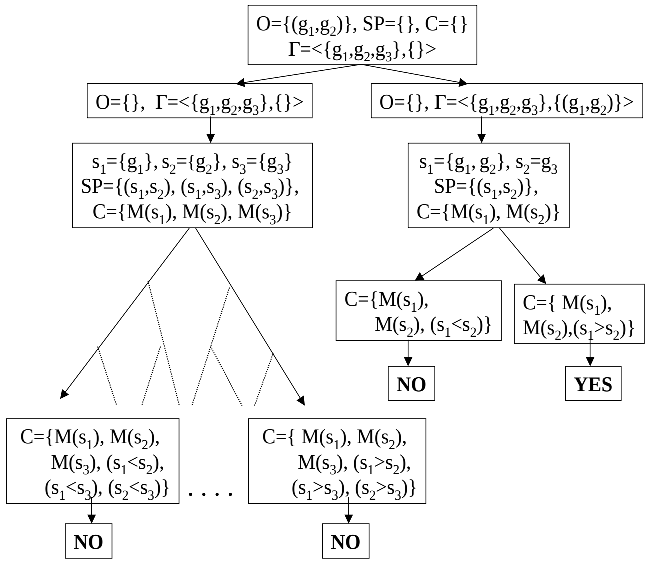
Figure 1 shows the execution trace of the algorithm SolveConstraints (O, Γ,SP,C) *when it is invoked by* CheckConsistency(G) *with the parameters shown in the root node. In the left subtree,* SolveConstraints *explores the cases where g_1 and g_2 do not overlap and Γ contains 3 connected components. Each leaf of the left subtree corresponds to a different possible ordering of the connected components; none of these leaves have solvable constraints. In the right subtree* SolveConstraints *explores the cases where g_1 and g_2 do overlap and Γ contains 2 connected components. A solution exists when the combined motion of g_1 and g_2 comes after g_3.*

Theorem 2 *Algorithm* Consistent(G) *is correct, i.e., G is consistent iff there is a way to make the nondeterministic choices in Step 2 such that the algorithm returns "yes." Likewise, Algorithm* CheckConsistency(G) *is correct, i.e. G is consistent iff* CheckConsistency(G) *returns "yes."*

A class of *go*-theories called **separated *go*-theories** can be defined for which the problem of checking consistency is polynomially solvable. In addition, given an arbitrary *go*-theory G, it is possible to check whether G is separated or not in polynomial time. For details, see (Yaman, Nau, & Subrahmanian 2004).

Answering in() Queries

In this section, we show how to check whether an atom of the form $a = \mathsf{in}(o, x_1, y_1, x_2, y_2, t_1, t_2)$ is a logical consequence



O={(g1,g2)}, SP={}, C={} Γ=<{g1,g2,g3},{}>
O={}, Γ=<{g1,g2,g3},{}>
O={}, Γ=<{g1,g2,g3},{(g1,g2)}>
s1={g1}, s2={g2}, s3={g3} SP={(s1,s2), (s1,s3), (s2,s3)}, C={M(s1), M(s2), M(s3)}
s1={g1, g2}, s2=g3 SP={(s1,s2)}, C={M(s1), M(s2)}
C={M(s1), M(s2), (s1<s2)}
C={ M(s1), M(s2),(s1>s2)}
NO
YES
C={M(s1), M(s2), M(s3), (s1<s2), (s1<s3), (s2<s3)}
C={ M(s1), M(s2), M(s3), (s1>s2), (s1>s3), (s2>s3)}
. . . .
NO
NO

Figure 1: Trace of CheckConsistency(G) for Example 2

of a *go*-theory G. Let us use $Rec(a)$ to denote the rectangle whose lower left corner is (x_1, y_1) and whose upper right corner is (x_2, y_2). For the sake of simplicity suppose for now that G contains just one go-atom g. The following lemma is not difficult to show.

Lemma 1 *Suppose $G = \{g\}$ is a* go-*theory and $a = \mathsf{in}(o, x_1, y_1, x_2, y_2, t_1, t_2)$ is a ground* go-*atom. a is a logical consequence of g iff the following two conditions hold:*

- *$Rec(a)$ must intersect the line segment $(loc_1(g), loc_2(g))$;*
- *During the time interval $[t_1, t_2]$, the object o must be in L, where $L = ((x'_1, y'_1), (x'_2, y'_2))$ is the sub-segment of $(loc_1(g), loc_2(g))$ that intersects $Rec(a)$).*

Before we can use this intuition to design an algorithm to check if a is a logical consequence of an arbitrary *go*-theory, we need one more definition.

Definition 1 *Suppose G is a set of* go-*atoms and s is a connected component of the graph Γ associated with G by algorithm* Consistent(G). *Then*

1. *The* extent *of s is given by the interval*
$$[\min\{t_1^-(g) \mid g \in s\}, \max\{t_2^+(g) \mid g \in s\}].$$
2. *s can* possibly overlap *$a = \mathsf{in}(o, x_1, y_1, x_2, y_2, t_1, t_2)$ iff $extent(s) \cap [t_1, t_2] \neq \emptyset$.*

When attempting to determine if a is a logical consequence of G, we also need to make sure that for every nondeterministic trace of algorithm Consistent(G), the constraints C force $a = \mathsf{in}(o, x_1, y_1, x_2, y_2, t_1, t_2)$ to be true. This is because each solution of the constraints C of algorithm Consistent(G) corresponds to a set of interpretations that satisfies G.

Let S be a connected component of the graph Γ described in algorithm Consistent(G) and $L = [P1, P2]$ be the union of the line segments in set $\{loc_1(g), loc_2(g) \mid g \in S\}$. Let

$\{p_1, \ldots, p_n\}$ be the union of $\{loc_1(g_i), loc_2(g_i) \mid g \in S\}$, with the points given in ascending order of their distance from P_1. For any point P which is on the line segment L, the earliest time at which object o could arrive at point P can be found by solving the following linear programming problem, $LPmin(C, S, d)$:

1. **I**f $P = p_i$ for some i and T_i is the variable associated with p_i in $Movement(S)$ then $LPmin(C, S, d)$ is the following linear programming problem:

   minimize T_i **subject to** C

2. **I**f the previous case does not apply and P is on line segment $[p_i, p_{i+1}]$ for some i and T_i is the variable associated with p_i in $Movement(S)$ then $LPmin(C, S, d)$ is the following linear programming problem:

   minimize $T_i + dist(P, p_i)/v_i^+$ **subject to** C

Another linear programming problem, $LPmax(C, S, P)$, is defined with exactly the same constraints as above, except that we replace v_i^+ with v_i^- and maximize the objective function rather than minimizing it. The result is the latest time at which the object could arrive at the point P.

```
Algorithm CheckIn($G, \Gamma, C, a$)
Suppose $a = \mathsf{in}(0, x_1, y_1, x_2, y_2, t_1, t_2)$;
$S$ = set of all connected components of the graph $\Gamma$
        associated with $G$ in algorithm Consistent($G$)
$C$ = set of constraints associated with $G$ and $\Gamma$ in
        algorithm Consistent($G$)
for each connected component $s \in S$ do
if $s$ can overlap $[t_1, t_2] \wedge LS(s) \cap Rec(a) \neq \emptyset$ then
        Suppose $LS(s) \cap Rec(s)$ has $P_1, P_2$ as its
        end points;
        Let $T_{min}$ be the solution of $LPMin(C, s, P_1)$
        Let $T_{max}$ be the solution of $LPMax(C, s, P_2)$
        if $T_{min} \geq t_1$ and $T_{max} \leq t_2$
        then return true and halt
        else continue
end for
return false
```

The following theorem says that CheckIn is correct as long as the *go*-theory is consistent.

Theorem 3 *Suppose G is a consistent* go-*theory and $a = \mathsf{in}(o, x_1, y_1, x_2, y_2, t_1, t_2)$ is a ground atom. Then: a is a logical consequence of G iff for every Γ,and C associated with G in algorithm* Consistent(G), *algorithm* CheckIn(G, Γ, C, a) *returns "true" when C is solvable.*

Example 3 *Suppose we have a go theory G=$[g_1, g_2, g_3]$ for the object o, and let $a = \mathsf{in}(o, x_1, y_1, x_2, y_2, t_1, t_2)$. Figure 2 contains a rectangle $R(a)$ and three lines ℓ_1, ℓ_2, ℓ_3 representing the movements defined by g_1, g_2 and g_3 respectively. Furthermore, suppose a trace of* Consistent(G) *algorithm requires that we do g_1 first, then g_2, and finally g_3. If we assume $[t_1, t_2]$ is wide enough so that all g_i's temporally overlap, then the algorithm* CheckIn *will perform the following checks:*

- *o arrives at P_1 after t_2: then answer is NO*

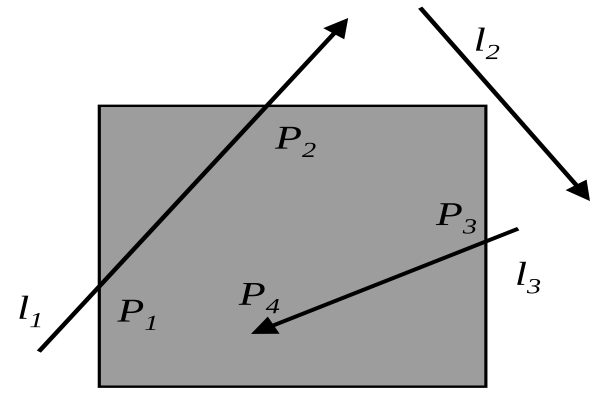


Figure 2: grapical representation of G and $R(a)$ in Example 3

- *o arrives at P_1 before t_2 and*
  - *o arrives at P_2 after t_1: then answer is YES*
  - *o arrives at P_2 before t_1 and*
    - *o arrives at P_3 after t_2: then answer is NO*
    - *o arrives at P_3 before t_2 and*
      - *o arrives at P_4 after t_1: then answer is YES*
      - *o arrives at P_4 before t_1: then answer is NO*

Answering near() Queries

In this section, we show how to check whether an atom of the form $b = \mathsf{near}(o, o', d, t_1, t_2)$ is a logical consequence of a *go*-theory G. The basic idea of the algorithm is as follows.

1. Let $\mathsf{space(o,t)}$ denote the set of points in space at which object o could be at time t. (We will define this term more explicitly later in this section).
2. We want to check if there exists a $t \in [t_1, t_2]$, an $(x_1, y_1) \in \mathsf{space(o,t)}$ and an $(x_2, y_2) \in \mathsf{space(o',t)}$ such that the distance between (x_1, y_1) and (x_2, y_2) exceeds d. If so, we return "no" (b cannot possibly be a logical consequence of G) - otherwise we return "yes."

This means we are left with the problem of defining $\mathsf{space(o,t)}$ for any object o and any time t, given a *go*-theory G.

Consider a *go*-theory G with one go-atom g. Obviously we only know where the object will be in the interval $TI(g) = [t_1^+(g), t_2^-(g)]$. During this interval, the object will be somewhere in the line segment $LS = L[loc_1(g), loc_2(g)]$. Furthermore for any given time $t \in TI(g)$, we can compute a smallest subsegment $L = [P_1, P_2]$ of LS such that the object can be anywhere on L and still satisfy the temporal constraints defined by g. We call L the **space envelope** of the object at time t.

Example 4 *Let*

$$g = go(obj, 40, 0, 40, 50, 10, 15, 16, 22, 5, 10).$$

The hexagonal area in Figure 3 is the space envelope of g for every $t \in [10, 22]$. The two dimensions of the figure

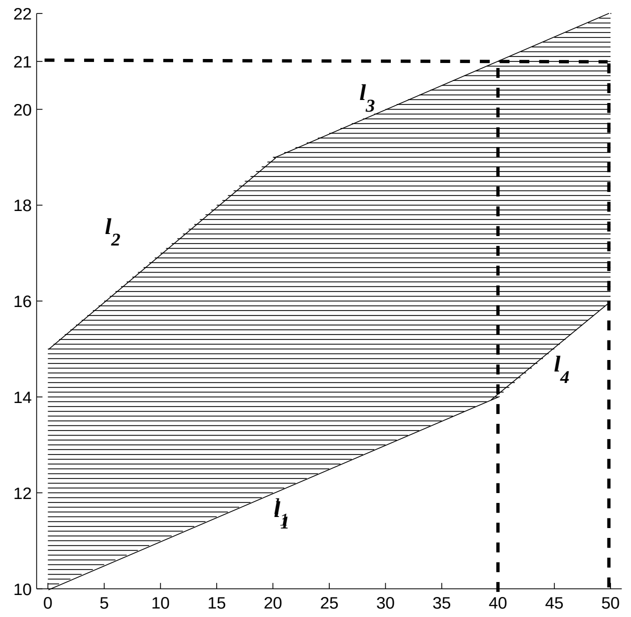


Figure 3: Graphical representation of space envelope in Example 4

are time and the y coordinate of obj (we omit the x dimension because it is constant). The area shows where on the line $(40, 0, 40, 50)$, *obj can be without violating to temporal and speed constraints defined in g. The four points on this graph* $(10, 0)$, $(15, 0)$, $(16, 50)$ *and* $(22, 50)$ *correspond to the positions at the earliesst /latest departure/arrival times. As seen in the figure there are four lines l_1 (bottom left), l_2 (top left), l_3 (top right), and l_4(bottom right) that define the boundaries of this hexagonal shape. Both l_1 and l_4 have a slope of 10, the fastest possible speed. Both l_2 and l_3 have a slope of 5, the slowest possible speed. The bold dashed lines show the space envelope of obj at time 21, which is* $[40, 50]$.

The next paragraph explains how to answer a query of the form $\mathsf{near}(\ldots, t_1, t_2)$ in the case where $t_1 = t_2$ (i.e., the time interval consists of just one time point). The subsequent paragraph extends this approach to the case where $t_1 \neq t_2$.

Suppose we have a *go*-theory G containing two go-atoms g_1 and g_2 for objects o_1 and o_2 respectively and we want to know if the distance between o_1 and o_2 is at most d at time t. Let's assume t is in $TI(g_1)$ and $TI(g_2)$. In this case, we can predict the positions of o_1 and o_2 at t. Let $L = [P_1, P_2]$ and $L' = [P'_1, P'_2]$ be the space envelopes of o_1 and o_2 at t. Then the answer is "yes" if the maximum distance between line segments L and L' is at most d. The maximal distance between the lines $L = [P_1, P_2]$ and $L' = [P'_1, P'_2]$ is given by $\max\{dist(P_i, P'_j) \mid 1 \leq i \leq 2, 1 \leq j \leq 2\}$ and can be computed in constant time.

In order to generalize this approach for a time interval $[t_1, t_2]$, all we need to do is represent each of the expressions P_1, P_2, P'_1 and P'_2 as a function of time and use constraints to check if $\max\{dist(P_i, P'_j) \mid 1 \leq i \leq 2, 1 \leq j \leq 2\} > d$ for any $t \in [t_1, t_2]$. If such a t exists, then the answer is "no"

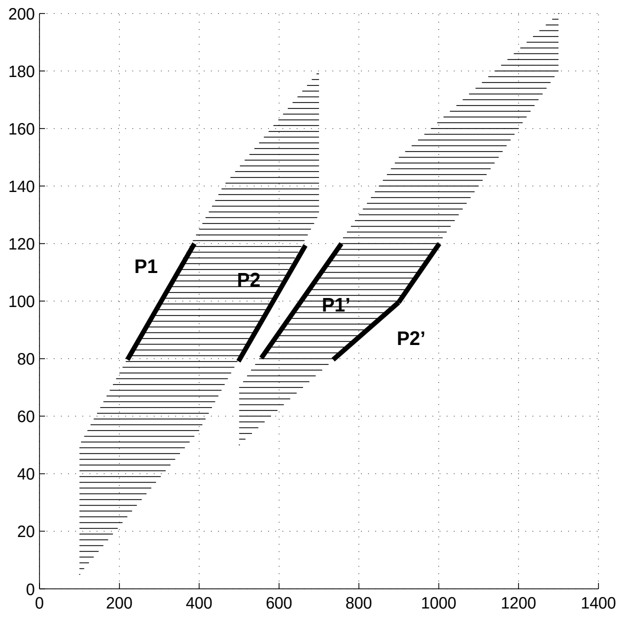


Figure 4: Graphical representation of space envelopes in Example 5

(because this means that there is at least one time instance, viz. t at which the two objects are more than d units apart from each other according to one interpretation) otherwise the answer is "yes." Note that as long as $P_i(t)$ and $P'_j(t)$ are expressible as piecewise linear functions, solving the above inequalities takes linear time.

Example 5 *Let*

$$g_1 = go(o_1, 100, 100, 100, 700, 5, 50, 130, 180, 4, 6);$$

$$g_2 = go(o_2, 100, 500, 100, 1300, 50, 70, 180, 200, 5, 8).$$

Let $b = \mathsf{near}(o, o', 580, 80, 120)$. *Figure 4 shows the space envelops for obj_1 and obj_2 along the y axis. Also the piecewise linear functions P_1, P_2, P'_1 and P'_2 in the interval* $[80, 120]$ *are displayed on the figure. Looking at the figure we can conclude that if o_1 is on P_1 and o_2 is on P'_2 the objects are fartest apart. Furthermore there are times when the distance is greater than 580, for example at time 120. Hence b is not a logical consequence of the* go*-theory $G = \{g_1, g_2\}$. We could however say that* $\mathsf{near}(o, o', 700, 80, 120)$ *is a logical consequence of G.*

We now extend this intuition for arbitrary *go*-theories. When attempting to determine if $b = \mathsf{near}(o, o', d, t_1, t_2)$ is a logical consequence of G, we need to make sure that for every nondeterministic trace of algorithm Consistent(G), the constraints C described in algorithm Consistent(G) force $b = \mathsf{near}(o, o', d, t_1, t_2)$ to be true. Given a *go*-theory G, let $\Gamma(o)$ and $C(o)$ denote the graph and the constraints associated with object o by algorithm Consistent(G). Suppose s is a connected component of $\Gamma(o)$ and it defines a movement on the line segment $[P_1, P_2]$ then we can define the following linear programming problem $NewLPmax(C, s)$ as follows:

maximize T **subject to** C, where T is the variable associated with point P_1 in $Movement(s)$. The solution to $NewLPmax(C, s)$ is the latest departure time for object o from P_1.

Another linear programming problem $NewLPmin(C, s)$ can be defined in a similar way except that in $NewLPmax(C, s)$ T is the variable associated with P_2 and the objective function minimizes T rather than maximizing it. The solution to $NewLPmin(C, s)$ is the earliest arrival time to P_2.

Definition 2 *Let G be a* go-*theory and b =* $\mathsf{near}(o, o', d, t_1, t_2)$ *be a ground atom. Let $\Gamma(o)$ and $C(o)$ denote the graph and the constraints associated with object o by algorithm* Consistent(G). *Suppose s is a connected component of* $\Gamma(o)$. *s is* temporally relevant to *b iff $[NewLPmax(C, s), NewLPmin(C, s)]$ contains the interval $[t_1, t_2]$.*

Note that given $\Gamma(o)$ and $C(o)$ there can be at most one component s that is temporally relevant to b. This is because each trace of the algorithm Consistent(G) checks whether a totally ordered connected components of Γ has a solution.

The following algorithm uses the functions $P_1(o, s, t), P_2(o, s, t)$ which return the closest and furthest points that o can be at time t according to the go-atoms in s. The full version of this paper contains the formal definitions for $P_1(o, s, t)$ and $P_2(o, s, t)$. Furthermore it also shows that for a given o and s, $P_i(o, s, t)$ is a piecewise linear function.

```
Algorithm CheckNear($G, \Gamma(o), C(o), \Gamma(o'), C(o'), b$)
Suppose $b = \mathsf{near}(o, o', d, t_1, t_2)$
$s$= connected component of $\Gamma(o)$ that is temporally
    relevant to $b$
$s'$= connected component of $\Gamma(o')$ that is temporally
    relevant to $b$
if $s$ or $s'$ is empty then return $false$
if $\exists t \mid dist(P_1(o, s, t), P_1(o', s', t)) > d$
        and $t_1 \leq t \leq t_2$ then return $false$
if $\exists t \mid dist(P_1(o, s, t), P_2(o', s', t)) > d$
        and $t_1 \leq t \leq t_2$ then return $false$
if $\exists t \mid dist(P_2(o, s, t), P_1(o', s', t)) > d$
        and $t_1 \leq t \leq t_2$ then return $false$
if $\exists t \mid dist(P_2(o, s, t), P_2(o', s', t)) > d$
        and $t_1 \leq t \leq t_2$ then return $false$
return $true$
```

The next theorem says that CheckNear is correct whenever the *go*-theory is consistent.

Theorem 4 *Suppose G is a consistent* go-*theory and b =* $\mathsf{near}(o, o', t_1, t_2, d)$ *is a ground atom. Then: b is a logical consequence of G iff for every $\Gamma(o)$,$C(o)$,$\Gamma(o')$ and $C(o')$ associated with G in algorithm* Consistent (G) *such that $C(o)$ and $C(o')$ are solvable, then algorithm* CheckNear*($G, \Gamma(o), C(o), \Gamma(o'), C(o'), b$) returns "true."*

Implementation and Experiments

We have built a prototype system called LOM which allows us to represent and query go theories. The implementation includes 35 functions and 1256 lines of Matlab code. We have performed the tests on a mobile Athlon XP 1800 processor with 256 MB memory running Windows XP. Figure 5 shows computation time of four types of queries. We generated *go*-theories with the following properties: all points are in the rectangle $[1, 1, 500, 600]$ and the maximum speed allowed is 50. Four query templates were used:

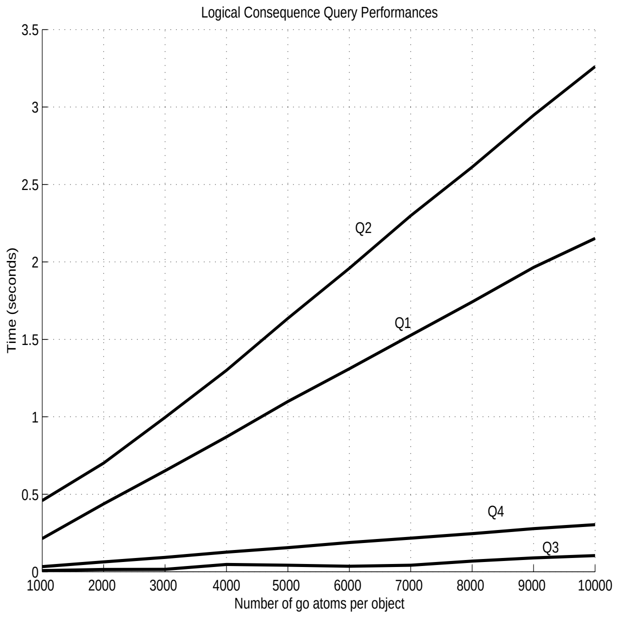


Figure 5: Performance of logical consequence queries

Q1: $in(o_1, 1, 1, 500, 600, 0.5*horizon, 0.75*horizon)$;
Q2: $in(o_1, 100, 150, 200, 350, horizon - 300,$
$horizon - 10)$;
Q3: $near(o_1, o_2, 50, 10, 12)$;
Q4: $near(o_1, o_2, 60, horizon - 20, horizon - 10)$;

where horizon is the latest arrival time for o_1 in the given theory. In each of these cases, we varied the value of horizon. The reader will notice that the implementation performs very well, executing most queries in a very small amount of time even when there are as many as 10,000 go-atoms per object.

Related Work

Some of the related work is by researchers in AI and philosophy who have studied *spatio-temporal logics*. These logics extend temporal logics to handle space. Most of them (Gabelaia *et al.* 2003; Merz, Zappe, & Wirsing 2003; Wolter & Zakharyaschev 2000; Cohn *et al.*), involve logical languages similar to LTL in which time is a discrete sequence of instants rather than being continuous.The focus of these works is qualitative - in contrast our work is heavily continuous and rooted in a mix of geometry and logic rather in just logic alone.

Despite many work on qualitative spatio-temporal theories (for a survay see (Anthony G. Cohn 2001)), very little work has been done on motion in such frameworks. (Muller 1998a; 1998b) describes a formal theory for reasoning about motion in a qualitative frame work. The expressive power of the theory allows for the definition of complex motion classes. The work however is purely symbolic hence has a different nature than our work. (Shanahan 1995) discusses the frame problem, when constructing a logic-based calculus for reasoning about the movement of objects in a real-valued co-ordinate system. Other than continuity the work does not specify any characteristics of the motion. (Rajagopalan & Kuipers 1994) focusus on relative position and orientation of objects with existing methods for qualitative reasoning in a newtoninan framework.

The database literature includes work moving-object databases, from the point of view of developing data structures to specify what objects are where and what their current velocity is and developing algorithms to answer queries projecting where the vehicles will be in the future (Chomicki & Revesz 1997; Erwig *et al.* 1999; Su, Xu, & Ibarra 2001). There are many differences between these works and ours: They do not provide a formal model theory, and they do not worry about consistency because they always record observed information about where the vehicles were observed in the past (presumably such a theory is consistent because if the vehicle was at one location at time t and another location at time t' there was a physical way for it to get from the first location to the second). They do not allow uncertainty (e.g., about starting times, ending times, and velocities).

Conclusions

In numerous applications there is a critical need to reason about moving objects. We have described a formal *logic of motion*, including a formal syntax and model theory.

Our work is rooted in a mix of geometry and logic: it provides a realistic continuous model of motion based on Newtonian physics. Our *go*-theories can represent uncertainty about starting times, ending times, and velocities. This ability is useful in the real world, where specifying exact velocities and exact arrival times is almost impossible.

We provide algorithms to check consistency; these algorithms are important because we want not only to reason about where vehicles were in the past, but where we expect them to be in the future. We show that consistency of *go*-theories is NP-complete. We also provide algorithms to answer two kinds of queries. in queries ask to find all vehicles that are guaranteed to be inside a given rectangular region at some time point in a given time interval. Such queries are very useful - for example, in the US Navy submarine tracking example mentioned in the introduction, we may want to find the identities of all enemy submarines that are guaranteed to be within a given region at a given point in time. Likewise, in the US Army's Combat Information Processor example mentioned in the introduction, we may want to find all enemy vehicles within a given region at a given point in time, in order to assemble resources to neutralize the threat. near queries in contrast ask for all pairs of objects that are guaranteed to be within a given distance of each other at some time point in a given interval. Again, this is very useful. In our US Navy route collision avoidance example,we want to know if two ships are scheduled to be very close to each other at some point in time. The same is true in air control traffic applications. Though we have not shown how our framework can be extended to complex formulas (involving conjunctions, disjunctions and negations), this is straightforward and will be included in the journal version of our paper.

An extremely important observation is that our algorithms for computing in and near queries run in polynomial time. This provides theoretical grounds for claiming that our methods are efficient. However, to validate this experimentally, we have also built a prototype implementation and shown that for a small sample of queries, our algorithms are extremely efficient. For systems containing upto 10,000 *go*-atoms about a single object, we are able to answer in queries in about 2-3 seconds. In contrast, we can answer near queries in under half a second with the same number of *go* atoms. These are very efficient.

We are currently extending the work in this paper in the following directions.

- We are examining the incorporation of new atoms (e.g. atoms to reflect the fact that a given object *may possibly* be within a given region in a given time interval) into our logic.
- For logic purists, we are developing a Hilbert-style proof theory.
- More importantly, we are studying classes of *go*-theories that have even better computational properties than those described in this paper.
- We are developing a scalable LOM implementation that allows us to conduct far more detailed experiments than those described here.

Acknowledgements

This work was supported in part by the Army Research Lab under contract DAAL0197K0135, the CTA on Advanced Decision Architectures, by ARO contract DAAD190010484, by Air Force Research Laboratory contract F30602-00-2-0505, by Naval Research Lab contract N00173021G005 and by NSF grants IIS0222914 and 0205489.

The opinions expressed in this paper are those of authors and do not necessarily reflect the opinions of the funders.

References

Anthony G. Cohn, S. M. H. 2001. Qualitative spatial representation and reasoning: An overview. *Fundam. Inform.* 46(1-2):1–29.

Chomicki, J., and Revesz, P. Z. 1997. Constraint-based interoperability of spatiotemporal databases. In *Proceedings of the 5th International Symposium on Large Spatial Databases (SSD)*, volume 1262, 142–161.

Cohn, A. G.; Magee, D.; Galata, A.; Hogg, D.; and Hazarika, S. Towards an architecture for cognitive vision using qualitative spatio-temporal representations and abduction.

In Freksa, C.; Habel, C.; and Wender, K., eds., *Spatial Cognition III*, Lecture Notes in Computer Science. Springer-Verlag. to appear.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

Ellis, R., and Gulick, D. 1978. *Calculus with Analytic Geometry*. New York: Harcourt Brace Jovanovich.

Erwig, M.; Güting, R. H.; Schneider, M.; and Vazirgiannis, M. 1999. Spatio-temporal data types: An approach to modeling and querying moving objects in databases. *GeoInformatica* 3(3):269–296.

Gabelaia, D.; Kontchakov, R.; Kurucz, A.; Wolter, F.; and Zakharyaschev, M. 2003. On the computational complexity of spatio-temporal logics. In Russell, I., and Haller, S., eds., *Proceedings of the 16th AAAI International FLAIRS Conference*, 460–464. AAAI Press.

Garey, M., and Johnson, D. 1977. Two-processor scheduling with start-time and deadlines. *SIAM J. Cornput* 6:416–426.

Karmarkar, N. 1984. A new polynomial time algorithm for linear programming. *Combinatorica* 4(8):373–395.

Khachiyan, L. G. 1979. A polynomial time algorithm for linear programming. *Soviet Math. Dokl.* 20(1):191–194.

Lloyd, J. 1987. *Foundations of logic programming*. Springer-Verlag.

Merz, S.; Zappe, J.; and Wirsing, M. 2003. A spatio-temporal logic for the specification and refinement of mobile systems. In Pezzè, M., ed., *Fundamental Approaches to Software Engineering (FASE 2003)*, volume 2621 of *Lecture Notes in Computer Science*, 87–101. Warsaw, Poland: Springer-Verlag.

Muller, P. 1998a. A qualitative theory of motion based on spatio-temporal primitives. In Cohn, A. G.; Schubert, L.; and Shapiro, S. C., eds., *KR'98: Principles of Knowledge Representation and Reasoning*, 131–141. San Francisco, California: Morgan Kaufmann.

Muller, P. 1998b. Space-time as a primitive for space and motion. In *FOIS'98*, 63–76.

Rajagopalan, R., and Kuipers, B. 1994. Qualitative spatial reasoning about objects in motion: Application to physics problem solving. In *IEEE Conf. on AI for Applications*, 238–245.

Shanahan, M. 1995. Default reasoning about spatial occupancy. *Artificial Intelligence* 74(1):147–163.

Su, J.; Xu, H.; and Ibarra, O. H. 2001. Moving objects: Logical relationships and queries. In *Proceedings of the Seventh International Symposium on Spatial and Temporal Databases*, 3–19.

Wolter, F., and Zakharyaschev, M. 2000. Spatio-temporal representation and reasoning based on RCC-8. In Cohn, A. G.; Giunchiglia, F.; and Selman, B., eds., *KR2000: Principles of Knowledge Representation and Reasoning*, 3–14. San Francisco: Morgan Kaufmann.

Yaman, F.; Nau, D.; and Subrahmanian, V. 2004. A foundational logic of motion. Technical report, Computer Science Dept., University of Maryland. To appear.