On Moving Objects in Dynamic Domains

Fangzhen Lin*
Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong

Abstract

In the physical world, an object is a moving object if its position changes over time. In a symbolic dynamic system such as a computer program or a blocks world, what are the moving objects? In this paper, we propose a definition, consider ways to generate moving objects and their "positions". We also introduce the flow graph of a moving object. It depicts the "trajectory" of the object, thus should be useful when planning to achieve a goal that involves moving some objects around.

Introduction

In the physical world, an object is a moving object if its position changes over time. The notion is clear here because we have a global time and a global spatial coordinate. The same cannot be said about symbolic dynamic systems often found in computer science and artificial intelligence, and it is by no means obvious what the moving objects and their positions are in such dynamic systems. For example, consider a domain where the robot can paint some walls into various colors. Depending on the axiomatization, the objects in this domain could be the robot, the walls, and the colors. Now which of them are moving, and what are their positions? Of course, there are domains where the notion of moving objects is clear. Consider the logistics domain [Bacchus, 2001] where there are packages that need to be delivered from one location to another, and there are trucks for moving packages inside a city and airplanes for moving them to a different city. Here obviously airplanes, trucks and packages are moving objects while locations and cities are not.

Why are we interested in knowing whether an object is a moving one or not? First of all, information about objects that can move or be moved and how they move or can be moved from one location to another is crucial to understanding a dynamic domain. In fact, in many dynamic domains, they could be the only thing that one cares about.

The next question is where such information comes from. Most action description formalisms used in AI do not have a mechanism for specifying this information. One could consider adding such mechanism to these action description formalisms. Alternatively, one can consider ways to discover such information automatically from an action domain description. One can argue that perhaps the former is the right way to go as the axiomatizers should have this information to begin with. While this may be the case, an effective way of discovering such information is still useful. At the very least, it would provide a way to check whether the user's specification agrees with the result from the automatic approach.

In this paper, we propose a formal definition of moving objects, and consider how information about them can be computed.

*This work was supported in part by HK RGC under GRF 616208.

Transition systems

To abstract away action formalisms, we consider transition systems. Given a first-order language L, a transition system is $(\mathcal{M}, T)$, where $\mathcal{M}$ is a set of first-order structures in L, and T a binary relation. Intuitively, $\mathcal{M}$ is the set of legal states and T represents the transition function on the legal states. Typically, structures in $\mathcal{M}$ should share the same domain of objects, but this is not required.

The problem that we are considering is then given such a transition system $(\mathcal{M}, T)$, and an object a in the domain of a structure in $\mathcal{M}$, whether a is a mobile object, and if yes what its positions.

Basic definitions

First, some notations. If $\varphi(x_1, ..., x_n)$ is a formula with distinct free variables $x_1, ..., x_n$, and $a_1, ..., a_n$ objects in the domain of a structure M, then we write $\varphi(a_1, ..., a_n)$ to denote the expression of replacing a_i for every free occurrences of x_i in φ for all $1 \leq i \leq n$. Notice that formally speaking, $\varphi(a_1, ..., a_n)$ is not a formula. For any such expression $\varphi(a_1, ..., a_n)$, we write $M \models \varphi(a_1, ..., a_n)$ if for some distinct variables $x_1, ..., x_n$ not occurring in $\varphi(a_1, ..., a_n)$, and some variable assignment σ such that $\sigma(x_i) = a_i, 1 \leq i \leq n$, $M, \sigma \models \varphi(x_1, ..., x_n)$ ($\varphi(x_1, ..., x_n)$ is true in M under σ), where $\varphi(x_1, ..., x_n)$ is the result of replacing a_i by x_i in $\varphi(a_1, ..., a_n)$ for all $1 \leq i \leq n$.

In the following, if P is a predicate and $u = (a_1, ..., a_n)$

a tuple of objects that matches the arity of P, then we call $P(u)$ a property of a_i, for every $1 \leq i \leq n$.

Definition 1 *Let $\mathcal{M}$ be a non-empty set of structures that have the same domain, and a an element of the domain. We say that a is mobile in $\mathcal{M}$ if there are $k \geq 2$ properties $p_1, ..., p_k$ about a such that*

- *for each i, there is an $M' \in \mathcal{M}$ such that*

$$M' \models p_i. \tag{1}$$

- *for each $M' \in \mathcal{M}$ and any $i \neq j$,*

$$M' \models p_1 \vee \cdots \vee p_k, \tag{2}$$
$$M' \models \neg(p_i \wedge p_j). \tag{3}$$

Informally, $p_1, ..., p_k$ are the k different "positions" of object a. Condition (1) means that each of these positions is possible, and (2) and (3) says that in every state, a is in exactly one of these positions. Notice that we require k to be greater than 1 for otherwise (3) would be trivial. To be specific, in the following, we will sometimes say that a is mobile in $\mathcal{M}$ under the properties $p_1, ..., p_k$.

The following result is immediate:

Proposition 1 *If a is mobile in $\mathcal{M}$ under a set A of properties, then for each $p \in A$ there are two states M and M' in $\mathcal{M}$ such that $M \models p$ and $M' \models \neg p$.*

Given a transition system $(\mathcal{M}, T)$, and a state $M \in \mathcal{M}$, we write $T(M)$ for the set of states reachable from M:

- $M \in T(M)$.
- If $S \in T(M)$, and $(S, S') \in T$, then $S' \in T(M)$.
- $T(M)$ is the smallest set that satisfies the above two conditions.

Definition 2 *Let $(\mathcal{M}, T)$ be a transition system, $M \in \mathcal{M}$, and a an element in the domain of M. We say that a is mobile in M under $p_1, ..., p_k$ if a is mobile in $T(M)$ under $p_1, ..., p_k$.*

In many cases, we want to know if an object is mobile in the whole transition system, not just in one particular state.

Definition 3 *An object is mobile in a transition system if it is mobile in every state of the system, and it is non-mobile if it is not mobile in every state.*

If an object is mobile in one state, and not mobile in another, then it is neither mobile nor non-mobile in the system.

Let's see some examples.

In the following, we shall represent a state M (a structure of a first-order language) as the set of atoms that are true in M. Thus $M \models P(u)$ iff $P(u) \in M$.

Given a transition system $(\mathcal{M}, T)$, we have been calling elements of $\mathcal{M}$ states of the system. Sometimes, we also call them legal states to emphasize the difference between states in the transition system and arbitrary models or sets of atoms.

The blocks world

Consider classical blocks world with fluents on, $ontable$, $holding$, $handempty$, and $clear$, and actions $pickup$, $putdown$, $stack$, and $unstack$.

For this domain, the transition function is actually not relevant because $T(M)$ is the same as the set of all states, i.e. from any given state, one can reach any other state through a sequence of actions.

Suppose there are just two blocks a and b. There are only a handful of legal states:

$$S_1 = \{handempty, ontable(a), on(b,a), clear(b)\},$$
$$S_2 = \{handempty, ontable(b), on(a,b), clear(a)\},$$
$$S_3 = \{handempty, ontable(b), ontable(a), clear(a), clear(b)\},$$
$$S_4 = \{holding(a), ontable(b), clear(b)\},$$
$$S_5 = \{holding(b), ontable(a), clear(a)\}.$$

Consider block a, the sets of mutually exclusive properties of a are as follows:

$$\{ontable(a), on(a,b), holding(a)\}$$
$$\{on(b,a), clear(a), holding(a)\}$$

The first one says that a is either on the table, on another block or held by the robot. The second one says that a is either clear, held by the robot or under another block. These are two alternative ways of describing the "positions" of a block.

Now consider a version of the blocks world with Fl a special constant denoting the floor [Nilsson, 1998]. With this constant, the fluent $ontable$ is no longer needed. With two blocks a and b, the set of legal states is as follows:

$$S'_1 = \{handempty, on(a, Fl), on(b,a), clear(b), clear(Fl)\},$$
$$S'_2 = \{handempty, on(b, Fl), on(a,b), clear(a), clear(Fl)\},$$
$$S'_3 = \{handempty, on(b, Fl), on(a, Fl), clear(a), clear(b), clear(Fl)\},$$
$$S'_4 = \{holding(a), on(b, Fl), clear(b), clear(Fl)\},$$
$$S'_5 = \{holding(b), on(a, Fl), clear(a), clear(Fl)\}.$$

The sets of mutually exclusive properties for blocks a and b are still the same as before, with $ontable(a)$ replaced by $on(a, Fl)$. The properties about Fl are $on(a, Fl)$, $on(b, Fl)$, $clear(Fl)$. They are not pair-wise mutually exclusive, thus Fl is not a moving object in any state.

The above analysis works for domains with more than two blocks as well. In fact, an analysis along the above line can be worked out on a general domain. This is what we are going to do for the logistics domain.

Logistics domain

The logistics domain [Bacchus, 2001] uses a multi-sorted language with the following primitive sorts:

$$truck, airplane, package, airport, location, city.$$

It also uses some defined sorts:

- $vehicle$ - the union of $truck$ and $airplane$,
- $physobj$ - the union of $package$ and $vehicle$,
- $place$ - the union of $airport$ and $location$.

The fluents are $inCity$ of arity $place * city$, at of arity $physobj * place$, and in of arity $package * vehicle$.

The actions are $loadTruck$, $loadAirplane$, $unloadTruck$, $unloadAirplane$, $driveTruck$, and $flyAirplane$. The assumption is that one can fly an airplane between any two airports, and drive a truck between any two locations within the same city (so a truck cannot go from one city to another).

There are too many legal states to enumerate here even for a domain with only a couple of objects for each primitive sort. However, we can consider each object in turn.

Consider a city object c. The set of properties about it is

$$\{inCity(l, c) \mid l \text{ a place}\}.$$

Since the truth values of these properties do not change from state to state, there cannot be any set of them on which c is mobile. Thus c is not a mobile object in any state.

Consider a location l. The set of properties about it is

$$\{inCity(l, c) \mid c \text{ a city}\} \cup$$
$$\{at(x, l) \mid x \text{ a package or a truck}\} \cup$$
$$\{at(apn, l) \mid apn \text{ an airplane}\}$$

The properties from the third set can never be true, and the truth values of those from the first set do not change over states, thus l can only be mobile on the properties from the second set. But the properties from the second set that are possible (satisfies (1)) are not mutually exclusive. Thus l is not a mobile object in any state.

Similar analysis would show that an airport is not a mobile object in any state.

Consider a truck t. The properties about t are

$$\{in(p, t) \mid p \text{ a package}\} \cup \{at(t, l) \mid l \text{ a place}\}.$$

Let M be a state, and $M \models at(t, l) \wedge inCity(l, c)$ for some place l (location or airport) and city c. There are two cases.

- If l is the only place such that $inCity(l, c)$ holds, then every $M' \in T(M)$ will satisfy $at(t, l)$ and for some packages $p_1, ..., p_k$, $in(p_i, t)$. Thus t is not mobile in M.
- If there is another place $l' \neq l$ such that $inCity(l', c)$ holds in M, then t is mobile: t is mobile on the following pair-wise mutually exclusive properties:

$$\{at(t, l') \mid M \models inCity(l', c)\}$$

The case for an airplane is similar: it is mobile if there are more than one airport, not otherwise.

Consider a package p. The set of properties about it are

$$\{in(p, v) \mid v \text{ a vehicle}\} \cup \{at(p, l) \mid l \text{ a place}\}$$

These properties are mutually exclusive. Thus they make p mobile if at least two of them are true in some of the states in $T(M)$. A state where p is not mobile is when p is at a location in a city that does not have any trucks.

Wall painting

The notion of "mobile" or "moving" studied here is very general. "Changing" may be a better word.

Consider a domain with walls and colors, and the action of painting a wall into certain color: $paint(w, c)$ always succeeds and changes the color of the wall w to color c.

So there are two sorts here, $wall$ and $color$, and one fluent $color(w, c)$. The set of properties about a wall w_0 is

$$\{color(w_0, c) \mid c \text{ a color}\}.$$

Assuming that a wall can have only one color in any initial state, then all future states will have the same property. Thus the above properties about w_0 are mutually exclusive, one of them must be true, and each of them is true in some of the states because of the painting action. So w_0 is a mobile object in any state, and its "position" is its color. However, a color is not a mobile object because properties about it are not mutually exclusive: two different walls can have the same color.

One reviewer questioned whether it is intuitive to say that a wall is a moving object. Naturally, whether something is intuitive depends on one's perspective. What we consider here are abstract transition systems. If we change "walls" to "blocks", "colors" to "locations", and "paint(w,c)" to "move(w,c)", we get an isomorphic system where as expected blocks are moving objects and locations are not.

Discussions

1. In our definition of a mobile object, we consider properties about the object to be atomic formulas that mention the object. One question is why not consider arbitrary formulas that mention the object. One difficulty with arbitrary formulas is that there is an infinite number of them. Another is that it would not work. For instance, the conditions in the definition would be trivially true if we let p_1 to be $P(c)$ and p_2 to be $\neg P(c)$, as long as there is an action that can change the truth value of $P(c)$.
2. Whether an object is mobile according to our definition clearly depends on the language. Consider the blocks world domain. If we add a new fluent $non(x, y)$, meaning that x is not on y, then we effectively make formula $\neg on(x, y)$ an atomic one. This, among other things, would make Fl mobile as $on(a, Fl)$ and $non(a, Fl)$ would be mutually exclusive and one of them is true in every state.
3. Our notion of mobile objects is very closely related to that of legal states. We assume here that the set of legal states is given as part of a transition system. Often, the legal states are specified by constraints like (2) and (3) [Lin, 2004].

Computing moving objects

When the number of states is large, finding out if an object is mobile by exploring the entire transition graph is not feasible. However, we can concentrate on just the part of the states that are relevant to the properties of the given object, as we have done in the logistics domain.

Let M be a state of L, and A a set of atoms of the form $P(u)$, where P is a predicate of L, and u a tuple of objects in the domain of M. We write M_A, the restriction of M on A, as

$$M_A = \{P(u) \mid P(u) \in M \cap A\}.$$

If $\mathcal{M}$ is a set of structures, then $\mathcal{M}_A$ is $\{M_A \mid M \in \mathcal{M}\}$.

Lemma 1 *Let $\mathcal{M}$ be a structure, F a set of properties about an object a, and $P_1, ..., P_k \in F$. We have that a is mobile in $\mathcal{M}$ under $P_1, ..., P_k$ iff it is mobile in $\mathcal{M}_F$ under $P_1, ..., P_k$.*

Proof: Straightforward. ■

Given a set $\mathcal{M}$ of states, and a set F of properties about a, the following non-deterministic function $mobile(\mathcal{M}, F)$ returns a subset $P \subseteq F$ under which a is mobile, if such a P exists, and $\emptyset$ otherwise.

By the above lemma, we assume that $\mathcal{M} = \mathcal{M}_F$.

Function $mobile(\mathcal{M}, F)$.

1. Let
$$X_1 = \{p \mid p \in F \wedge \neg\exists M.M \in \mathcal{M} \wedge p \in M\},$$
$$X_2 = X_1 \cup (\bigcap \mathcal{M}),$$
$$F' = F \setminus X_2,$$
$$\mathcal{M}' = \mathcal{M}_{F'}.$$
2. Let $P_0 = \{p \mid \{p\} \in \mathcal{M}'\}$. (If the function returns a non-empty set, this set must contain P_0.)
3. If there are two distinct $p, p' \in P_0$ such that $\{p, p'\} \subseteq M$ for some $M \in \mathcal{M}'$, then return $\emptyset$. (In this case, p and p' are not mutually exclusive, but must both in P, a contradiction. So a is not mobile in this case.)
4. $X = \mathcal{M}'$, $P = P_0$, $Q = \emptyset$.
5. for each $M \in X$, if $M \cap P \neq \emptyset$ then $Q = Q \cup M$ and delete M from X;
6. The following is a loop with backtracking

   while $X \neq \emptyset$ do
   choose a p such that
   $\exists M.M \in X \wedge p \in M \setminus Q$ and $\forall M.M \in X \wedge p \in M \rightarrow M \cap Q = \emptyset$; $P = P \cup \{p\}$; backtrack if no such p, and return $\emptyset$ if cannot backtrack; for each $M \in X$, if $M \cap P \neq \emptyset$ then $Q = Q \cup M$ and delete M from X;
   return P;

Let's run this function on the blocks world with two blocks. The full set of legal states is $\mathcal{S} = \{S_1, ..., S_5\}$, and for a,
$F = \{clear(a), ontable(a), on(a,b), on(b,a), holding(a), on(a,a)\}$.

- $\mathcal{S}_F$ is:
$$S_{11} = \{ontable(a), on(b,a)\},$$
$$S_{12} = \{on(a,b), clear(a)\},$$
$$S_{13} = \{ontable(a), clear(a)\},$$
$$S_{14} = \{holding(a)\}.$$
So by the above lemma, we use this set as $\mathcal{M}$ to run the function.
- Step 2 return with $F' = F$. So $\mathcal{M}' = \mathcal{M}$.
- Only S_{14} is a singleton in $\mathcal{M}$, so $P_0 = \{holding(a)\}$.
- Step 4: $X = \{S_{11}, S_{12}, S_{13}, S_{14}\}$, $P = \{holding(a)\}$, and $Q = \emptyset$.
- Step 5: $X = \{S_{11}, S_{12}, S_{13}\}$, $P = \{holding(a)\}$, $Q = \{holding(a)\}$.
- The loop. Four possible choice of p in the first iteration: $ontable(a), on(a,b), on(b,a), clear(a)$.
  - Choosing $p = ontable(a)$ yields
  $$P = \{ontable(a), holding(a)\},$$
  $$X = \{S_{12}\},$$
  $$Q = \{ontable(a), holding(a), on(b,a), clear(a)\},$$
  and then in the next iteration $p = on(a,b)$, and then outputs $P = \{ontable(a), holding(a), on(a,b)\}$.
  - Choosing $p = on(a,b)$ yields
  $$P = \{on(a,b), holding(a)\},$$
  $$X = \{S_{11}, S_{13}\},$$
  $$Q = \{on(a,b), holding(a), clear(a)\},$$
  and then in the next iteration, there will be two choices for p: $ontable(a)$ or $on(b,a)$. Choosing $p = ontable(a)$ will make $P = \{ontable(a), on(a,b), holding(a)\}$, $X = \emptyset$, thus outputs $P = \{ontable(a), on(a,b), holding(a)\}$, the same one as before. Choosing $p = on(b,a)$ will yield $P = \{on(b,a), on(a,b), holding(a)\}$, $X = \{S_{13}\}$, $Q = \{ontable(a), on(b,a), on(a,b), holding(a), clear(a)\}$. This means that there won't be any possible choice for p in the next iteration, thus causing backtracking.
  - Choosing $p = on(b,a)$ yields
  $$P = \{on(b,a), holding(a)\},$$
  $$X = \{S_{12}, S_{13}\},$$
  $$Q = \{on(b,a), holding(a), ontable(a)\},$$
  and then in the next iteration, there will be two choices for p: $clear(a)$ or $on(a,b)$. Choosing $p = on(a,b)$ will cause backtracking, and choosing $p = clear(a)$ will yield an output $P\{clear(a), on(b,a), holding(a)\}$.
  - Choosing $p = clear(a)$ will yield
  $$P = \{clear(a), holding(a)\},$$
  $$X = \{S_{11}\},$$
  $$Q = \{clear(a), holding(a), ontable(a)\}.$$
  There will then only be one choice of p in the next iteration: $p = on(b,a)$, which will then output $P = \{on(b,a), clear(a), holding(a)\}$.

In general, if the blocks world domain has n blocks, then the set of properties about a block a will be

$$A = \{holding(a), clear(a), ontable(a)\} \cup \{on(a,b) \mid b \text{ a block}\} \cup \{on(b,a) \mid b \text{ a block}\}$$

which is linear in n. Let $\mathcal{M}$ be the set legal states. We know that the size of $\mathcal{M}$ is exponential in n. However, the size of $\mathcal{M}_A$ is linear in n: S is in $\mathcal{M}_A$ iff

- $S = \{holding(a)\}$ or
- $S = \{ontable(a), clear(a)\}$ or
- $S = \{ontable(a), on(b, a)\}$ for some $b \neq a$ or
- $S = \{on(a, b), clear(a)\}$ for some $b \neq a$ or
- $S = \{on(a, b), on(c, a)\}$ for some $c \neq b \neq a$.

While in general the above procedure for $mobile(\mathcal{M}, F)$ may run in exponential time because of backtracking, it runs in $O(n^2)$ in the blocks world, and has two outputs: $P = \{holding(a), ontable(a), on(a, b_1), ..., on(a, b_{k-1})\}$ or $P = \{holding(a), clear(a), on(b_1, a), ..., on(b_{k-1}, a)\}$, where $b_1, ..., b_{k-1}$ are the other blocks that are different from a.

Flow graphs

We have considered the definition of moving objects and how to compute them. We now consider how properties about them can be used in planning.

Let M be a structure in a transition system $(\mathcal{M}, T)$, a an object that is mobile on the positions $p_1, ..., p_n$. The *flow graph* of a is a directed graph whose nodes are $p_1, ..., p_n$, and there is an edge from p_i to p_j, $i \neq j$, if there is a state S reachable from M such that for some transition $(S, S') \in T$, $S \models p_i$ and $S' \models p_j$. Notice that because of exclusiveness, $S \models \neg p_j$ and $S \models \neg p_i$.

The flow graph shows that to move a mobile object from one position, say p_i to a new position p_j, it must go through the intermediate positions on one of the paths in the graph from p_i to p_j.

In the blocks world, a flow graph for a block x is a cycle that has edges going out and into $holding(y)$ from each node in $\{ontable(x), on(x, y), on(x, z), ...\}$ (assuming that we use this set as the positions of a block).

In the logistics domain, for a truck, its flow graph is a complete graph, and similarly for an airplane. For a package p, there are two edges (one outgoing and one incoming) between $at(p, l)$ and $in(p, t)$ (t is a truck) if p and t are in the same city, and there are two edges between $at(p, l)$ and $in(p, apl)$ (apl is an airplane) if l is an airport.

We see that the flow graphs for blocks in the blocks world, and those for packages in the logistics domain are actually quite similar: in the former, $holding(x)$ is the node that connects the other positions, while in the latter positions of the form $in(p, x)$ are connecting points.

The flow graph of a moving object depicts how it can be moved from one location to another independent of other objects. Often we need to consider how more than one objects can be moved. For instance, we may want x to be on y, and y on the table; or that more than one packages to be delivered to various locations.

Consider two objects a and b. Suppose initially they are at p_1 and q_1, respectively, but need to be moved to p_2 and q_2, respectively. Either there is an action that can move a and b to their respectable goal position simultaneously or at some point, either a is moved to its goal position first and while keeping a where it is, b is moved to its goal position or the other way around. So we consider flow graphs under a constraint.

Let G be a state sentence. The flow graph a in a state M under the transition system $(\mathcal{M}, T)$ and constraint G is again a directed graph whose nodes are $p_1, ..., p_n$, and there is an edge from p_i to p_j, $i \neq j$, if there is a state S reachable from M such that for some transition $(S, S') \in T$, $S \models G \wedge p_i$ and $S' \models G \wedge p_j$.

Consider a blocks world with three blocks a, b, and c. The flow graph of a under the constraint $ontable(b)$ is the same as the flow graph of a without constraints, meaning that a's position is independent of the constraint $ontable(b)$. However, the flow graph of b under the constraint $on(a, b)$ has no edges as none of the actions about b can be performed given this constraint. This means that to achieve the joint goal $on(a, b) \wedge ontable(b)$, one should achieve $ontable(b)$ first for otherwise, once $on(a, b)$ is true, there is no way to achieve $ontable(b)$ without making $on(a, b)$ false.

Concluding remarks

We have defined a notion of moving objects and their "positions", and considered ways to compute them. In terms of application, if the objects in a planning domain can be divided into those that are moving according to our definition and those that are not, and furthermore those that are not do not change their properties from states to states, then flow graphs and flow graphs under constraints can be a powerful tool to help with planning in such domains.

References

Bacchus, F. 2001. AIPS'00 planning competition. *AI Magazine* 22(3):47–56. See also http://www.cs.toronto.edu/aips2000.

Lin, F. 2004. Discovering state invariants. In *Proceedings of the Nineth International Conference on Principles of Knowledge Representation and Reasoning (KR2004)*, 536–544.

Nilsson, N. J. 1998. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann Publishers, Inc., San Francisco, CA.