Position Systems in Dynamic Domains

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

December 22, 2012

Abstract

A dynamic domain consists of a set of legal states and a transition function that maps states to states. AI formalisms for specifying dynamic domains have so far focused on describing the effects of actions, that is, the transition functions. In this paper we proposed a notion of characteristic set of position systems for the purpose of describing legal states. A position system for a type of objects is a set of properties that are mutually exclusive, and that in each state, every object of the type must satisfy exactly one of these properties called its position under the position system. A set of position systems, one for each type of objects in the domain, is characteristic if there is a one-to-one mapping between legal states and sets of objects' positions under these position systems. We show that once we have a characteristic set of position systems for a dynamic domain, planning can be done by writing rules about when to move an object from one position to another.

1 Introduction

A dynamic domain or transition system consists of a set of states and a transition function that maps states to states. While this is a simple mathematical notion, using it in practice is a challenging task because the number of states is often large, and the transition functions may be complex. In fact, much of work in AI on reasoning about actions is about good formalisms for specifying transition systems. These formalisms include the situation calculus [McCarthy, 1968; Reiter, 2001], event calculus [Kowalski and Sergot, 1986], fluent calculus [Thielscher, 1998], action languages [Gelfond and Lifschitz, 1999; Giunchiglia and Lifschitz, 1998], and STRIPS-like languages [Fikes and Nilsson, 1971; McDermott, 1998]. However, all these formalisms are used primarily for specifying the effects of actions, that is, the transition functions.

Consider the blocks world domain. A STRIPS style specification of the domain would need to provide for each action three lists: the precondition list that specifies the conditions that must be satisfied for the action to be executed successfully, the add list that specifies the properties that will be made true by the action, and the delete list that specifies the properties that will be made false by the action. For instance, for action $stack(x, y)$, they are

- Precondition list: $holding(x)$, $clear(y)$.
- Add list: $handempty$, $on(x, y)$.

- Delete list: $holding(x)$, $clear(y)$.

This description works on states given by sets of atoms, those true in the states. It does not explicitly say which states are legal, and is not supposed to be meaningful when applied to an illegal state, for example, when the robot is holding two blocks at the same time. This is similar to that in logic, false implies everything. However, the user needs to make sure that an action description like the above one for $stack(x, y)$ will not yield an illegal state from a legal one.

The same is true for Reiter's successor state axioms, which are of them form:

$$F(x, do(a, s)) \equiv \varphi(x, a, s),$$

where $\varphi(x, a, s)$ is a formula about situation s. They are intended to specify how the truth values of $F(x)$ in the successor state $do(a, s)$ is to be determined using a formula about the current state s, provided that the current state s is legal and a is executable. If any of these conditions is not true, then while the situation $do(a, s)$ still exists and its corresponding state determined using the successor state axioms, their validity is not guaranteed. In other words, the successor state axioms are supposed to make sense only when the current state is legal and the action is executable. But again, which state is considered legal is not explicitly stated, and the user is supposed to understand it well enough so that starting from a legal state, the new state computed using the successor state axioms will also be legal.

There has been few work on specifying the space of legal states. One notable exception was a complete axiomatization of the legal blocks world states by Cook and Liu [2002]. On the other hand, the space of legal states is closely related to state constraints for which there has been an extensive body of work (e.g. [Finger, 1986; Ginsberg and Smith, 1988b; 1988a; Carbonell *et al.*, 1991; Lin and Reiter, 1994; Lin, 1995; 2004; Huang *et al.*, 2000]). Intuitively, a state is legal if it satisfies all the state constraints. But to our knowledge, there has not been any work on how to systematically write down these state constraints, which is what this paper is about.

Having an explicit representation of legal states is useful. Many have observed that state constraints can be used to prune the search space in planning (e.g. [Carbonell *et al.*, 1991; Huang *et al.*, 2000]). Many have even argued that a user should first write down explicitly the state constraints, then use these constraints to derive the indirect effects of actions (the so-called ramification problem) as well as implicit action preconditions (the so-called qualification problem) (e.g. [Ginsberg and Smith, 1988b; 1988a; Lin and Reiter, 1994; Lin, 1995]).

What we propose in this paper is to use a special class of constraints, what we call "position systems" to characterize legal states. Consider the logistics domain that consists of cities, locations, packages and vehicles. In any legal state, a package is either at exactly one of the locations or in exactly one of the vehicles, and a vehicle is at exactly one of the locations. Thus the properties "at location l" and "in vehicle h" are like the "positions" of packages, and "at location l" the positions of vehicles. We now proceed to make this precise.

2 Basic definitions

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.

We consider typed objects. The types can be represented as sorts in a many-sorted first-order language L or by unary predicates.

Let $\mathcal{M}$ be a set of states (structures in L). Let $\varphi_i(x, \mathbf{y_i})$ be a formula that may contain variable x of type τ and variables in $\mathbf{y_i}$ as free variables, and n_i either a natural number or $\inf$, $1 \leq i \leq k$. We say that $\{(\varphi_1(x, \mathbf{y_1}), n_1), ..., (\varphi_k(x, \mathbf{y_k}), n_k)\}$ is a position system in $\mathcal{M}$ for objects of type τ if the following properties are true in all states in $\mathcal{M}$:

- for every object x of type τ, one of the property $\varphi_i(x, \mathbf{y_i})$ must be true:

$$\forall x. \exists \mathbf{y_1} \varphi_1(x, \mathbf{y_1}) \vee \cdots \vee \exists \mathbf{y_k} \varphi_k(x, \mathbf{y_k}). \tag{1}$$

- each object of type τ can satisfy just one property:

$$\forall x, \mathbf{y_i}, \mathbf{z}. \varphi_i(x, \mathbf{y_i}) \wedge \varphi_i(x, \mathbf{z}) \supset \mathbf{y_i} = \mathbf{z}, \tag{2}$$

$$\forall x. \exists \mathbf{y_i} \varphi_i(x, \mathbf{y_i}) \supset \forall \mathbf{y_j} \neg \varphi_j(x, \mathbf{y_j}), \tag{3}$$

  where $i \neq j$, $\mathbf{z}$ is a tuple of new variables of the same size and type as $\mathbf{y_i}$, and $\varphi_i(x, \mathbf{z})$ the result obtained from φ_i by replacing the variables in $\mathbf{y_i}$ by their respective variables in $\mathbf{z}$.

- at most n_i objects of type τ can satisfy $\varphi_i(x, \mathbf{y_i})$, that is, for each i, if n_i is not $\inf$, then

$$\forall x_1, ..., x_{n_i+1}, \mathbf{y_i}. \varphi_i(x_1, \mathbf{y_i}) \wedge \cdots \wedge \varphi_i(x_{n_i+1}, \mathbf{y_i}) \supset (x_1 = x_2 \vee \cdots \vee x_1 = x_{n_i+1}). \tag{4}$$

Our first observation is that there are many position systems. For instance, trivially, $\{(P(x), \inf), (\neg P(x), \inf)\}$ is a position system. A good position system is one that can be used to capture legal states.

In the following, we make use of the following conventions. If $\varphi(x_1, ..., x_n)$ is a formula with a tuple of distinct free variables $(x_1, ..., x_n)$, and $(a_1, ..., a_n)$ a matching tuple of 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 $1 \leq i \leq n$.

If $\{(\varphi_1(x, \mathbf{y_1}), n_1), ..., (\varphi_k(x, \mathbf{y_k}), n_k)\}$ is a position system for objects of sort τ, M a legal state, and a an object of sort τ in the domain of M, then the position of a in M is the unique $\varphi_i(a, \mathbf{b})$ that is true in M, where $\mathbf{b}$ is a tuple of objects in M that match the free variables other than x in φ_i. In the following, we denote the position of a in M by $pos(a, M)$. If we have a position system for every sort in L, then every object in M will have a position. In the following, we denote by $pos(M)$ the set of positions of the objects in M:

$$pos(M) = \{pos(a, M) \mid a \text{ is an object in } M\}.$$

Given a transition system $(\mathcal{M}, T)$ in language L, a set $\mathcal{P}$ of position systems, one for each sort, is said to be characteristic for $\mathcal{M}$ iff for any two distinct states $M_1, M_2 \in \mathcal{M}$ that have the same domain, $pos(M_1) \neq pos(M_2)$. This means that a state in $\mathcal{M}$ can be represented by the set of objects and their positions in it.

The main message of this paper is that a characteristic set of position systems often exists, is easy to define, and useful. In the rest of the paper, we illustrate using three domains: the blocks world domain, the wall painting domain, and the logistics domain.

3 Examples

3.1 The blocks world

Consider the blocks world domain with fluents $on(x, y)$, $ontable(x)$, $holding(x)$, $handempty$, and $clear(x)$.

A state in the blocks world is legal if it contains a finite number of blocks, and has the property that among the blocks, at most one is held by the robot ($holding(x)$ is true), and the rest arranged into some non-overlapping single column towers on the table.

The following position system says that a block is either held by the robot, on the table, or on another block:

$$\{(on(x, y), 1), (ontable(x), \inf), (holding(x), 1)\}. \tag{5}$$

This system gives rise to the following axioms:

$$\forall x.\exists y on(x, y) \vee ontable(x) \vee holding(x), \tag{6}$$

$$\forall x, y_1, y_2.on(x, y_1) \wedge on(x, y_2) \supset y_1 = y_2, \tag{7}$$

$$\forall x.[\exists y on(x, y)] \supset \neg ontable(x), \tag{8}$$

$$\forall x.[\exists y on(x, y)] \supset \neg holding(x), \tag{9}$$

$$\forall x.ontable(x) \supset \neg holding(x), \tag{10}$$

$$\forall x_1, x_2, y.on(x_1, y) \wedge on(x_2, y) \supset x_1 = x_2, \tag{11}$$

$$\forall x_1, x_2.holding(x_1) \wedge holding(x_2) \supset x_1 = x_2. \tag{12}$$

The above set of axioms does not capture legal states. To do that, we need to add a second-order axiom to rule out cycles in the transitive closure of $on(x, y)$, and an axiom to rule out the possibility of the robot holding a tower of more than one blocks.

However, the position system (5) is characteristic. If two legal states have the same position for every blocks in their domain, then they are the same state.

Proposition 1 *If M_1 and M_2 are two legal blocks world states, and $pos(M_1) = pos(M_2)$, then $M_1 = M_2$.*

Thus a legal state in the blocks world can be represented by its set of object positions according to (5). For instance, the following two-blocks world

$$\{ontable(A), handempty, on(B, A), clear(B)\}$$

can be represented by the positions of the two blocks:

$$\{ontable(A), on(B, A)\}.$$

3.2 The wall painting domain

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. The legal states are those that assign exactly one color to each wall.

So there are two sorts here, $wall$ and $color$, and one fluent $color(w, c)$. For sort $wall$, the position system $\{color(x, y), \inf)\}$ yields the following axioms:

$$\forall x \exists y.color(x, y), \tag{13}$$

$$\forall x, y_1, y_2.color(x, y_1) \wedge color(x, y_2) \supset y_1 = y_2. \tag{14}$$

For sort $color$, there is no meaningful position system. In fact, a state is legal iff it satisfies (13) and (14). It follows that $\{color(x, y), \inf)\}$ for x is a characteristic position system.

3.3 Logistics domain

In logistics domain, the objects are packages, locations, cities, trucks, airplanes, vehicles, and airports. The class of airports is a subclass of locations, and a vehicle is either an airplane or a truck. There are two predicates: $in(x, y)$ (location x is in city y or package x is inside vehicle y), and $at(x, y)$ (package x or vehicle x is at location y).

Like the wall painting example, the legal states can be captured by the following position systems:

- For locations and airports, $\{(in(x, y) \wedge city(y), \inf)\}$.
- For packages, $\{(at(x, y) \wedge location(y), \inf), (in(x, y) \wedge vehicle(y), \inf)\}$.
- For airplanes: $\{(at(x, y) \wedge airport(y), \inf)\}$.
- For trucks: $\{(at(x, y) \wedge location(y), \inf)\}$. Notice that once a truck is in a city, it cannot be driven to another city. This is a constraint about dynamics of trucks, not on the legality of states.
- No meaningful position systems for cities.

4 Applications

For a dynamic domain, having a characteristic set of position systems is very useful. First of all, a position system yields a set of constraints that can be used to prune search space in planning (e.g. [Huang *et al.*, 2000; Etzioni, 1993; Carbonell *et al.*, 1991]). If we have a characteristic set of position systems, we can then represent states by the objects and their positions in the domain. This means that we can design a planner based on when to move which object to where. Assume that we have specifications for the predicate $moveTo(x, p_1, p_2, g)$: for goal g, if the position of x in the current state is p_1, then it needs to be moved to position p_2. Then a simple planner can work as follows:

```
while goal g is not true do
   choose an action A such that
     A is executable and
     for some x whose current position is p1, there is a position p2
     such that moveTo(x,p1,p2,g) can be proved and
     p2 holds after A is performed
```

We illustrate using the blocks world domain and the logistics domain.

4.1 The blocks world

It is easy to write an effective planner this way for the blocks world. In principle, for each block x and each position p, we need to specify the conditions under which x needs to be moved to position p. It is often convenient to do this by a sequence of Prolog-like rules.

We introduce an auxiliary predicate $noMove(x, g)$, meaning for goal g, the block x does not need to be moved. The following rules specify the predicate. The first two rules say that do not move a block that is

already part of a tower in the goal, and the last two rules say that do not move a block that is irrelevant for the goal. In the following, we capitalize variable names, and write $P(\mathbf{x})$ to mean that the atom $P(\mathbf{x})$ is true in the current state, and $P(\mathbf{x}, G)$ that the atom $P(\mathbf{x})$ is true in the goal state G.

$$noMove(X, G) \leftarrow ontable(X), ontable(X, G).$$
$$noMove(X, G) \leftarrow on(X, Y), on(X, Y, G), noMove(Y, G).$$
$$noMove(X, G) \leftarrow ontable(X), \neg holding(X, G), \neg ontable(X, G), \neg\exists Y.on(X, Y, G).$$
$$noMove(X, G) \leftarrow on(X, Y), noMove(Y, G), \neg holding(X, G), \neg ontable(X, G), \neg\exists Z.on(X, Z, G).$$

With this predicate, $moveTo$ is easy to specify. If $holding(x)$ is true for some block x in the current state, then if block x is ready to be moved to its goal position, then do so (first rule) else put it on the table (second rule). If $holding(x)$ is not true for any x, then pick a block x that is not forbidden to move and can be moved to position $holding(x)$ by an action in the current state. The following Prolog-like rules are sequentially ordered as they are written.

$$moveTo(X, holding(X), on(X, Y), G) \leftarrow on(X, Y, G), clear(Y), noMove(Y, G).$$
$$moveTo(X, holding(X), ontable(X), G).$$
$$moveTo(X, P, holding(X), G) \leftarrow \neg noMove(X, G).$$

This set of rules, when used by the forward chaining planner that was given earlier, behave in a way similar to TLP planner [Bacchus and Kabanza, 2000] and TAL planner [Doherty and Kvarnström, 2001].

Since (5) is a characteristic position system for the blocks world, a legal state can be represented by the set of blocks' positions in the state. Thus a state can be represented by a set of $O(n)$ atoms, where n is the number of blocks. Furthermore, given a set of blocks' positions, the truth value of other atoms like $clear(x)$ can be determined in linear time.

Proposition 2 *For the blocks world, assume that the initial state S is legal, complete and represented by $Pos(S)$, the set of the positions of the blocks in the state, and a goal state G is a set of atoms that are parts of a legal state, then the forward planner will find a plan of size at most $4n$ in polynomial time, where n is the number of blocks.*

Proof: Our first observation is that if $noMove(x, G)$ can be proved, then G can be achieved without any action being done on x. Our second observation is that if $holding(x)$ is true in the current state, then x is either moved to the table in the next state or to its goal position. Thus in the worst case, the algorithm will move all blocks on the table and build up the towers specified in the goal, and this involves at most $4n$ actions. When grounded, there are $O(n^2)$ possible actions, and $O(n^2)$ rules for $moveTo$ and $noMove$. Checking whether a rule can be fired takes time linear to the size of S and G, which is $O(n)$. Thus it takes $O(n^5)$ to decide on an action to do, and $O(n^6)$ to return a plan. This is a very rough analyses in the worst time. In practice, the algorithm should run much faster. ■

4.2 The logistics domain

Here we assume that a goal is a set of atoms of the form $at(p, l)$, where p is a package and l a location.

Again the following Prolog-like rules are interpreted sequentially in the order that are written. A rule can be applied only if none of the rules before it can be applied.

1. If package X is at an airport, load it to an airplane if it needs to go to another city, and load it to a truck if it needs to go to another location in the same city:

$$moveTo(X, at(X, L), in(X, V), G) \leftarrow$$
$$package(X), airport(L), airplane(V), at(X, L2, G), in(L, C1), in(L2, C2), C1 \neq C2.$$
$$moveTo(X, at(X, L), in(X, V), G) \leftarrow$$
$$package(X), airport(L), truck(V), at(X, L2, G), in(L, C), in(L2, C), L \neq L2.$$

2. If package X is at a location other than an airport, but needs to be moved, then load it to a truck:

$$moveTo(X, at(X, L), in(X, V), G) \leftarrow package(X), truck(V), \neg airport(L), at(X, L2, G), L \neq L2.$$

3. If package X is in a vehicle and the vehicle is at X's destination, then unload X from the vehicle:

$$moveTo(X, in(X, V), at(X, L), G) \leftarrow at(V, L), at(X, L, G).$$

4. If package X is in an airplane which is in the same city as X's destination is, then unload X from the airplane:

$$moveTo(X, in(X, V), at(X, L), G) \leftarrow airplane(V), at(V, L), at(X, L1, G), in(L, C), in(L1, C).$$

5. If package X is in a truck which is at an airport in a city that is not the same as X's destination is, then unload X from the truck so an airplane can pick it up:

$$moveTo(X, in(X, V), at(X, L), G) \leftarrow$$
$$truck(V), airport(L), at(V, L), at(X, L1, G), in(L, C), in(L1, C1), C \neq C1.$$

6. Fly an airplane Y to pickup package X if X is at an airport and needs to go to another city:

$$moveTo(Y, at(Y, L1), at(Y, L2), G) \leftarrow$$
$$airplane(Y), package(X), at(X, L2), airport(L2), at(X, L3, G), in(L2, C1), in(L3, C2), C1 \neq C2.$$

7. Drive a truck Y to pickup package X if it needs to go to another location:

$$moveTo(Y, at(Y, L1), at(Y, L2), G) \leftarrow$$
$$truck(Y), package(X), at(X, L2), in(L1, C), in(L2, C), at(X, L3, G), L2 \neq L3.$$

8. If a truck Y has a package X in it, then if X needs to be delivered to a location that is in the same city as the truck is, then drive Y to X's destination else drive the truck to an airport:

$$moveTo(Y, at(Y, L1), at(Y, L2), G) \leftarrow$$
$$truck(Y), package(X), in(X, Y), in(L1, C), in(L2, C), at(X, L2, G), L2 \neq L3.$$
$$moveTo(Y, at(Y, L1), at(Y, L2), G) \leftarrow$$
$$truck(Y), airport(L2), \neg airport(L1), package(X), in(X, Y), in(L1, C), in(L2, C).$$

The rules above for the logistics domain look more involved than the ones for the blocks world, but they are not really hard to write. Basically, we just go through all possible positions of a package and decide what to do.

We emphasize that the rules above are sequentially ordered. For example, the last rule says that if a truck is at a non-airport location L_1, contains a package, then it needs to go to an airport in the same city. This is correct only if the package's destination is not in the same city as the truck is. However, if this were the case, an earlier rule would have been applied first.

Similar to the blocks world domain, while this set of rules does not guarantee the planner to return shortest plans, it does make the planner run in polynomial time.

Proposition 3 *Assume that the initial state S is complete and represented by $Pos(S)$, and the goal G is a set of locations for packages. If the goal is achievable, then the forward planner will find a plan of size at most $12n$ in polynomial time, where n is the number of packages in the goal.*

Proof: For each package in the goal, it takes at most 12 actions to move it to its goal position: drive a truck to where it was, load it into the truck, drive the truck to an airport, unload it at the airport, fly an airplane to the airport, load the package into the airplane, fly the airplane to an airport in the city where the package is to be delivered, unload the package at the new airport, drive a truck to the new airport, load the package to the truck, drive the truck to the package's final location, and unload the package from the truck.

The results are easy to see by observing that according to the rules, the planner will not run into a cycle, and will not move any package that is not in the goal. ■

5 Concluding remarks

We have defined a notion of position systems, related it to state constraints and legal states. We have shown how it can be used in planning. Informally, a position system defines possible "positions" an object can have. Theoretically, there may be many position systems for a transition system. But the desired ones are those that are characteristic in the sense that the set of objects' positions according to these position systems identifies uniquely a legal state. We believe that most transition systems in AI have such characteristic position systems. One future work is to verify this by defining characteristic position systems for the known AI transition systems such as the planning domains used in the planning competitions (e.g. [Bacchus, 2001]).

5.1 Acknowledgments

This work grew out of my earlier work on moving objects [Lin, 2011], and I thank Ning Ding, Jianmin Ji, Maonian Wu, and Haodi Zhang for useful discussions. My special thanks to Jianmin for trying out ideas on using information about moving objects in planning, and his comments on an earlier version of the paper.

References

[Bacchus and Kabanza, 2000] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 16:123–191, 2000.

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

[Carbonell *et al.*, 1991] Jaime G. Carbonell, Oren Etzioni, Yolanda Gil, Robert Joseph, Craig A. Knoblock, Steven Minton, and Manuela M. Veloso. PRODIGY: An integrated architecture for planning and learning. *SIGART Bulletin*, 2(4):51–55, 1991.

[Cook and Liu, 2002] Stephen A. Cook and Yongmei Liu. A complete axiomatization for blocks world. *AMAI*, 2002.

[Doherty and Kvarnström, 2001] Patrick Doherty and Jonas Kvarnström. TALplanner: A temporal logic-based planner. *AI Magazine*, 22(3):95–102, 2001.

[Etzioni, 1993] Oren Etzioni. Acquiring search-control knowledge via static analysis. *Artificial Intelligence*, 62:255–301, 1993.

[Fikes and Nilsson, 1971] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to theorem proving in problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Finger, 1986] Jeff Finger. *Exploiting Constraints in Design Synthesis*. PhD thesis, Department of Computer Science, Stanford University, Stanford, CA, 1986.

[Gelfond and Lifschitz, 1999] Michael Gelfond and Vladimir Lifschitz. Action languages. *Electronic Transactions on Artificial Intelligence, http://www.ep.liu.se/ea/cis*, Vol 3, nr 016, 1999.

[Ginsberg and Smith, 1988a] Matthew L. Ginsberg and David E. Smith. Reasoning about action I: A possible worlds approach. *Artificial Intelligence*, 35:165–196, 1988.

[Ginsberg and Smith, 1988b] Matthew L. Ginsberg and David E. Smith. Reasoning about action II: the qualification problem. *Artificial Intelligence*, 35:311–342, 1988.

[Giunchiglia and Lifschitz, 1998] Enrico Giunchiglia and Vladimir Lifschitz. An action language based on causal explanation: Preliminary report. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI–98), AAAI Press, Menlo Park, CA.*, 1998.

[Huang *et al.*, 2000] Yi-Cheng Huang, Bart Selman, and Henry A. Kautz. Learning declarative control rules for constraint-based planning. In *ICML'2000*, pages 415–422, 2000.

[Kowalski and Sergot, 1986] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:6795, 1986.

[Lin and Reiter, 1994] Fangzhen Lin and Raymond Reiter. State constraints revisited. *Journal of Logic and Computation, Special Issue on Actions and Processes*, 4(5):655–678, 1994.

[Lin, 1995] Fangzhen Lin. Embracing causality in specifying the indirect effects of actions. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI–95), IJCAI Inc. Distributed by Morgan Kaufmann, San Mateo, CA.*, pages 1985–1993, 1995.

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

[Lin, 2011] Fangzhen Lin. On moving objects in dynamic domains. In *Tenth International Symposium on Logical Formalization of Commonsense Reasoning (Commonsense'2011)*, 2011.

[McCarthy, 1968] John McCarthy. Situations, actions and causal laws. In M. Minsky, editor, *Semantic Information Processing*, pages 410–417. MIT Press, Cambridge, Mass., 1968.

[McDermott, 1998] Drew McDermott. PDDL – the planning domain definition language. Technical Report TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998. ftp://ftp.cs.yale.edu/pub/mcdermott/software/pddl.tar.gz.

[Reiter, 2001] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press, 2001.

[Thielscher, 1998] M. Thielscher. Introduction to the fluent calculus. *Electronic Transactions on Artificial Intelligence*, 2(34):179192, 1998.