

Motion Planning in A Society of Mobile Agents

James Campbell and Albert Esterline
Department of Computer Science/NASA ACE
North Carolina A&T State University
Greensboro, NC 27400
Email: {jrc, esterlin}@ncat.edu

1. Introduction

Groups or societies of agents may collaborate on a common goal and thereby achieve more than by working individually. This paper discusses the use of societal theories to model the carrying out of a plan by a group of agents. We focus on plans that relate to agent motion since the low-level aspects of such plans are well understood and motion provides clear criteria for successful planning. We use statecharts to represent these plans, which are hierarchically structured and require concurrent behavior. We represent norms as statements in deontic logic, and we annotate statechart plans with such statements to indicate that certain norms apply to certain groups. In fact, the transitions in a statechart are often naturally interpreted in a normative way. Using deontic logic allows us to distinguish ideal behavior from actual behavior that violates norms and hence to represent fault tolerance. The follow section introduces societal norms and relates them to motion planning. Section 3 presents statecharts and introduces deontic statecharts. Section 4 is an extended example, and Section 5 discusses planning, specifically, how a successful multi-agent motion plan of the form we discuss may be constructed. Section 6 is the conclusion.

2. Societal Norms and Motion Planning

Central to social group theories is the idea of social norms [Tu95]. Social groups with structure have some sort of task or role structure, such as organizations, institutions and some teams. The members of these structured groups act on behalf of the group and are governed by means of formal or informal rules. The rules can be characterized in terms of social norms, which may be rules called r-norms or proper social norms called s-norms. Norms in a society or group are reasons for members of that group or society either to act or not to act, with sanctions applied accordingly based on permissions. The social position of a societal or group member has attached to it certain tasks, roles and rights. An r-norm task is what is required of a social position by virtue of an r-norm, such as the duties of a job. An example of an s-norm task is what is required in the role of a father-figure as specified by proper social norms, such as doing yard work.

It is critical to note that norms can be violated without being invalidated. For example, an employee may fail to fulfill the duties of his job and a father may fail in his role. The agent violating the norm, not the norm itself, is blamed. In contrast, if certain behaviors run counter to an empirical generalization, we reject that generalization as false and do not blame the agent.

Collision-free movement of an agent through an environment populated with obstacles is a major objective of motion-planning research [Fu91]. With multiple agents, the agents in a group or in multiple groups must work together in coordinating moves, and the size of the search space can increase exponentially. Societal norms governing how the agents collaborate can reduce the number of parameters, thereby pruning the search space. For example, with the appropriate norm, an agent would not need to decide whether to stay on the right or the left side of the road – the norm would be reason to perform the appropriate behavior. Also, communication between agents is reduced since the norms of the group dictate many of the aspects of the actions of each agent. Note that norms may be violated, often with serious consequences for the group endeavor. Generally, group members depend on the other members to follow the norms and apply some sanction to members violating the norms.

3. Statecharts

Statecharts [Ha87] are useful in modeling complex reactive systems. Statecharts are visual formalisms used to describe or model states and transitions. Statecharts extend conventional state-transition diagrams to include the ideas of concurrency, hierarchy and communication. Hierarchy gives the ability to encapsulate or to group substates into superstates. For example, Figure 1 shows a statechart with two states, D and B . State D has two substates, A and C . This configuration is an Exclusive Or (XOR) decomposition: if the system is in state D , then it is in either substate A or substate D (but not both). Suppose that the system is in state B . On event e , it goes directly to substate A of state D – see the arrow labeled “ e ”. On event h , it goes to state D as a whole – see the arrow labeled “ h ”, which stops at the boundary of D . But the unlabeled arrow to substate C indicates that C is the default start state of state D ; so, on event h in state B , it ends up going to C . Now suppose that the system is in state D (either

substate). On event f , it goes to state B – see the arrow labeled “ f ”, which starts at the boundary of D . Finally, suppose that the system is in substate A . On event g , it goes to substate C as long as condition S holds – see the “ S ” in parentheses after the label “ g ”.

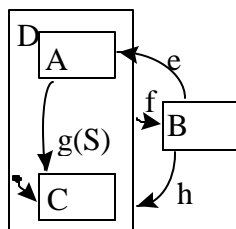


Figure 1.

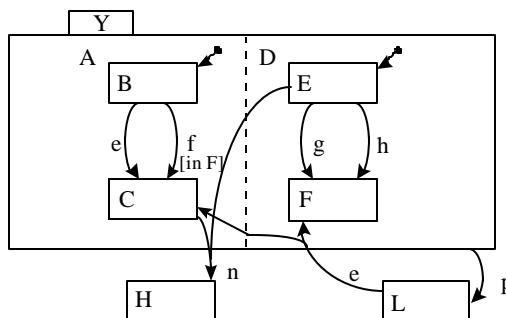


Figure 2.

Statecharts show concurrency through orthogonality. Orthogonality is an AND decomposition and is captured by the partition of the statechart as illustrated in Figure 2. For example, Figure 2 shows a statechart with three states, H , L , and Y . Y has two orthogonal components, A and D – see the dashed boundary. State A has two substates, B and C ; D has two as well, E and F . If the system is in Y , it must be in one of four possible pairs of substates (one from A , one from D); it could be in B and E , B and F , C and E , or C and F . Now suppose that the system is in state L . On event e , it enters substates C and F . If the system is in substate E (and either B or C), then, on event g , it goes to F (and stays in whatever substate of A it was in). If the system is in substate B (and either E or F), then, on event e , it goes to C . If the system is in B , it can go to C on event f if and only if it is also in substate F – see the “*in F*” in square brackets added to the label “ f ”. If the system is in substates C and E , then, on event n , it goes to state H . If the system is in state Y in any of the four ways mentioned above, then, on event p , it goes to state L .

There are a few enhancements to statecharts that we shall find useful. To begin with, we can add a time interval $[l,u]$ to the event labeling a transition [La95]. This indicates that the event must occur no earlier than l time units after the condition on the transition becomes true (or the source state is entered if there is no condition) and no later than u time units after this comes about. Similarly, we can append an interval $[l,u]$ to the action labeling a transition to indicate that the action can take no less than l and no more than u time units. Finally, we use T , possibly subscripted, as a label on a transition to indicate an internal event that triggers that transition. This is useful when an agent is free to determine when it will perform an action – as long as it is in an appropriate state.

The most significant enhancement is to introduce into statecharts deontic logic operators, enabling one to handle deviations from ideal behavior [Lu97]. Deontic Logic [MW93] is the logic of reasoning about ideal and actual behavior. The modal (specifically, deontic) operators in deontic logic express permission, prohibition, and obligation. Modal operators can modify a given proposition. For example, the proposition *Betty washes the clothes* could be modified using the obligation operator, thus forming the sentence *It is obligated that Betty washes the clothes*. In relation to norms, Betty’s role in the group could be maid. The duties of a maid include to wash clothes. In formal notation, O is the obligation operator; if we let p stand for the proposition *Betty washes the clothes*, then the above statement of obligation could be expressed as Op . There are also operators for permission (P) and prohibition (F). We can define any two of these operators in terms of the third; as is usual, we shall take obligation to be the primitive notion.

We use a version of deontic logic in which a deontic operator can be subscripted with the name of an agent to indicate that the obligation, permission, or prohibition applies to that agent. For example, if A is the name of some agent and p stands for “The book is returned,” then O_{Ap} means that A is obligated to see to it that the book is returned. The name could actually denote a group of agents. For example, let G denote a group of dockworkers and let p be the proposition “The pallet is lifted on the truck.” Then O_{Gp} asserts that the group of dockworkers has the obligation to see to it that the pallet is lifted on the truck. The case where a group as a whole has a certain obligation must be distinguished from the case where each member of the group, individually, has a certain obligation. By abuse of notation, we apply deontic operators, not only to propositions, but also to phrases denoting actions, giving propositions with the obvious meaning.

The example in the next section shows how we annotate statecharts with deontic statements. Once we allow deontic statements, we can generally interpret transitions deontically as well, except when the system represented is a natural, physical system with no intended purpose. Suppose, for example, that the behavior of a person is being

described. Then there is a general obligation that the person perform the actions and change state as indicated by the part of the statechart in question. Similarly, there is a general prohibition on performing an action when that action is not sanctioned by the statechart. Finally, there are obligations that relate to the time intervals added to events and actions labeling transitions. We allow that those obligations and prohibitions might be violated. A violation will lead to a violation state where wrongs are righted and the agent is penalized and possibly reformed before he returns to the normal states. If the agent is an artifact that fulfills a purpose, norm violation is faulty behavior, and violation states involve such things as rectifying adverse consequences of the fault, testing the agent, and repairing or adjusting the agent. Thus, deontic statecharts allow us to represent fault tolerance.

4. Example

We give an example in which a society of mobile agents follows a plan. Here a society of dockworkers has the collective goal to ship all requested pallets by the end of each day. This society of dockworkers is composed of two groups. Group *B* is responsible for inspecting and moving pallets from a storage location to a shipping and receiving location. Group *A* is responsible for receiving pallets and preparing them for shipment. Two moving obstacles, *Ca* and *Cb*, are in the environment of the dockworkers and may impede their paths. Referring to Figure 3, obstacle *Ca* is first at location 4, moves to location 5, then back to 4. *Cb* starts at location 6, moves to location 5, then back to 6. We assume that *Ca* and *Cb* are never at 5 at the same time. The dockworkers in group *B* include *Ba*, the boss, and loaders *Bb* and *Bc*. Their task is to move a pallet from location 8 to location 7, have it accepted by the receiver and his boss, and return to location 8 for another pallet. *Ba*, *Bb*, and *Bc* start at locations 1, 2, and 3, respectively. They first assemble at location 4. They then proceed to location 8, where the boss inspects the pallet and the loaders pick it up. The loaders, carrying the pallet, proceed to location 7 via location 5; they must wait, if necessary, for *Ca* or *Cb* to vacate location 5. At 7, *Ab* is waiting for the pallet, and *Aa* (his boss) comes from location 9 to 7 to inspect the pallet. If it passes the inspection, then *Ab* receives it and *Bb* and *Bc* leave the pallet and go to 10. There they wait (if necessary) for location 5 to become unoccupied so they may go via it to location 4, where the cycle begins again. If the pallet does not pass the inspection, then the loaders go back via the same locations but bring the pallet with them as far as location 8. When *Aa* is finished inspecting the pallet, he returns to location 9; *Ab* remains at location 7.

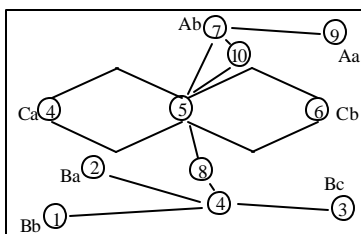


Figure 3.

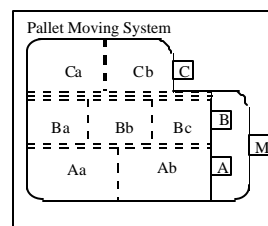


Figure 4.

Figure 4 shows the higher-level composition of the statechart of the pallet moving system. Dashed lines are repeated to show larger units of orthogonal composition. The system contains two orthogonal components, *C* and *M*, representing, respectively, the society of workers and the group of obstacles. Component *C* contains two orthogonal subcomponents, the moving obstacles *Ca* and *Cb*. Component *M* contains two subcomponents, *B* and *A*. *B*, the group of workers that moves the pallets, contains three orthogonal subcomponents, *Ba*, whose role is loader-boss, and *Bb* and *Bc*, both with the role of loader. *A*, the group of workers that receives the pallets, contains two orthogonal subcomponents, *Aa*, whose role is receiver-boss, and *Ab*, whose role is receiver. The duty of a loader agent in this society is to lift and carry pallets to a specified location. The duty of a loader-boss is to inspect a pallet before moving it and to coordinate the loader agents. The duty of a receiver agent is to receive pallets and to prepare necessary documentation for shipping. Finally, the role of a receiver-boss is to inspect and approve a pallet for transport and receipt.

Figure 5 shows the states and transitions for component *Bb* (a loader agent) from Figure 4. The states are grouped into two superstates: *Violation*, which contains as substates all the states that *Bb* enters as the result of a norm violation, and *Non-Violation*, which contains the remaining, “normal” states. The names of the normal states all begin with the name of the agent, “*Bb*”, so that similar agents may have similar states, distinguished from these only by the prefixes of their names. The next part of the name of a normal state indicates the location of *Bb* when it is in that state. If there is more than one state corresponding to a given location, then lowercase letters are used as suffixes to supply enough distinct names. For example, *Bb* is at location 8 when it is in either *Bb8a* or *Bb8b*.

The general flow is as follows. From *Bb1*, *Bb* goes to *Bb4*. Once *Ba* and *Bc* are also at location 4, *Bb* goes to state *Bb8* (picking up the pallet) then (carrying the pallet) to *Bb7*, where the pallet is inspected. If the pallet is approved, it is received, *Bb* goes to location 10 (*Bb10a*) to wait for the obstacles to clear, and then goes back to the staging state, *Bb4*, where the cycle begins again. If the pallet is not approved, then *Bb* again goes to the staging location, now indicated by state *Bb10b* (since we must distinguish the case where a pallet is being carried from the case where one is not), and stops off at location 8 – to leave the pallet – on the way back to the staging state, *Bb4*. Every action labeling a transition between normal states is a *move*. The various maximum and minimum times for the moves are indicated. In the “delivery” sequence, from *Bb1* to *Bb7*, transitions are assumed to be triggered by a *signal* event from the loader-boss. A *received* event (from the receiver agent) starts the return to *Bb4* without carrying a pallet, while a *not received* event starts the return while carrying the pallet to location 8. The events triggering the return transitions once state *Bb7* is left are all internal events, indicated by a subscripted “*T*”; the maximum time (as well as the minimum time) for the *T* event to happen is 0.

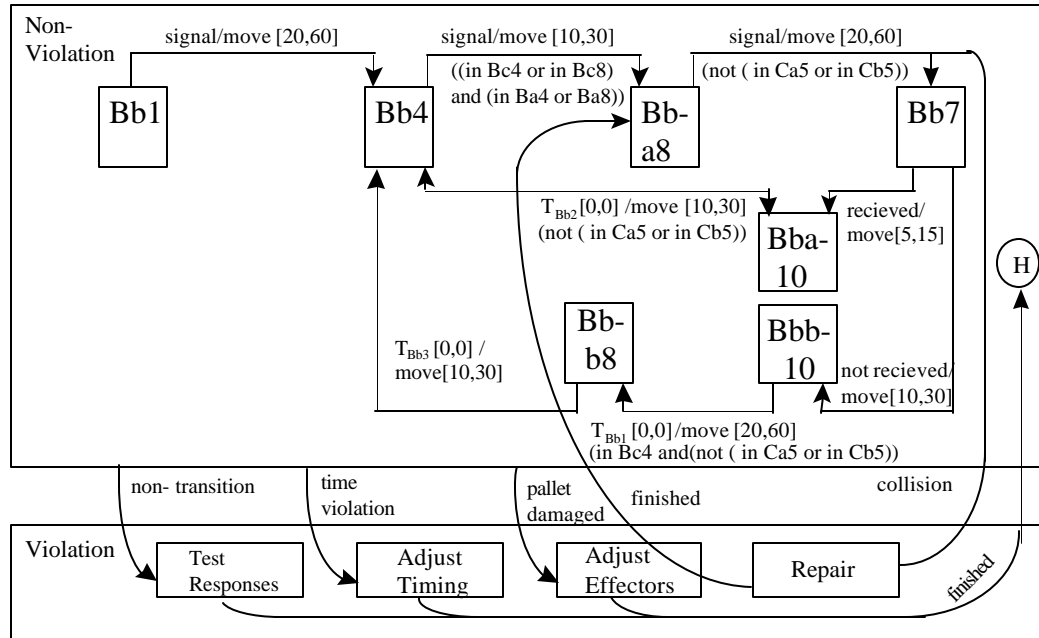


Figure 5

Each transition in Figure 5 involves several obligations. For example, consider the transition from state *Bb4* to state *Bb8*. The statechart indicates that, when the event *signal* happens, the component goes from state *Bb4* to state *Bb8*. We can paraphrase this in terms of Figure 3 as: when the signal is given, *Bb* ought to move from location 4 to location 8. This is an obligation. All of *Bb*'s violations of obligations to make transitions are handled uniformly: *Bb*'s responses are tested. This is indicated in Figure 5 by the arrow labeled “non-transition” from the boundary of the non-violation state to the *Test Responses* substate of the *Violation* state. *Test Responses* itself will have substates sufficient to express the steps of the testing procedure. The arrow labeled “finished” from *Test Responses* to the *H* in the non-violation state indicates that, when *Bb* is in the *Test Responses* state and the *finished* event happens (the tests have been completed and any adjustments have been made), it goes back to the state where the violation occurred. This will be the source state of the transition that failed. With agent *Bb* now properly adjusted, when event *signal* next occurs, it should indeed make the required transition.

Referring again to the transition from *Bb4* to *Bb8* in Figure 5, we see that the action *move* ought to be completed in no less than 20 sec. (any faster would be dangerous) and no more than 60 sec. (any slower would be undue delay). Violation of this obligation is taken to be a *time violation* event. Such an event can occur whenever any *move* action is undertaken. Figure 5 shows that this leads to the *Adjust Timing* violation state. Exit from this state (as from the *Test Responses* state) is an event *finished* and again goes back to where the violation occurred (see the arrow to *H*). Here the state we return to is the destination of the transition. The violation was in the time it took to get there, so the adjusted agent resumes from that state.

Several of the transitions in Figure 5 are labeled with conditions requiring that other components be in certain states. For example, the transition from *Bb8a* to *Bb7* has the condition *not(in Ca5 or in Cb5)*. This is violated if either of the mobile obstacles (*Ca* or *Cb*) is at location 5, through which *Bb* must pass in going from location 8 (state

Bb8a) to location 7 (state *Bb7*). So this violation is a *collision* event. It results in the transition from *Bb8a* to *Bb7* on event *signal* becoming a transition to the violation state *Repair* on the event pair *signal-collision* (in that order). The transition from *Repair*, labeled *finished*, is back to state *Bb8a*, so the agent can try again to go from location 8 to location 7. Note that here the violation does not occur at either the source state or the destination state, but in the course of the transition between the two. Any other condition could be violated, and this component of the statechart could become quite cluttered with violation transitions. An alternative to representing each such violation with an arrow in the diagram is to tabulate the violations outside that statechart. In such a table, there would be no need to invent names (such as *collision*) for condition violations.

The states and transitions for component *Bc* will be just like Figure 5 but with “*Bb*” and “*Bc*” exchanged throughout. The states and transitions for the loader-boss, *Bc*, will have the same structure as in Figure 5 for the delivery sequence from location 1 to location 7 since, even though *Ba* does not go beyond location 8, it must signal to the loaders to go to 7. The return for *Ba* is simply moving from 8 back to 4. The triggering events for *Ba* can all be internal events since it controls the moves of group *B*. *Ba*’s actions in the delivery sequence must include *signal* events that trigger transitions by *Bb* and *Bc*.

As an example of a norm that is associated with a state or component, not a transition, consider the obligation of the entire society *M* of agents that all pallets are shipped by 5:00 PM. Violation leads to state *Run Late*. This applies to the entire society of agents. The individual agents do not have *Run Late* violation states. Rather, there is a *Run Late* violation state for the entire society *M*. This state could be exploited as follows. Suppose we conjoin the following to the condition on the transition from *Bb4* to *Bb8a*, where *time* is the local time and \prec is the total order on time points expressed by “earlier than”:

(*) $time \prec 4:55 \text{ PM}$ and *not in Run Late*

Suppose similar conditions are conjoined to the corresponding transitions of *Ba* and *Bc*. Next, suppose that we add a transition from *Bb4* to *Bb1* on event *signal* with the negation of (*) as condition, and similar transitions are added for *Ba* and *Bc*. Then, when the loaders are back at location 4, it is 4:55 PM or later, and *M* is not in the *Run Late* state, then everyone will go home (*Ba* to 2, *Bb* to 1, and *Bc* to 3). If, however, the loaders are back at 4, it is 4:55 PM or later, but *M* is in the *Run Late* state, then *Ba*, *Bb*, *Bc* will proceed, as before, to carry a pallet. If *M* is in the *Run Late* state, it will exit this state when the pallets are exhausted – the details do not concern us here (nor does how the pallets are delivered).

5. Applying Statecharts in Shortest-Path Motion Planning

So far, we have been concerned with carrying out plans, not with how to form plans, that is, not with planning. When using a finite-state representation (such as statecharts), it is critical to identify discrete state values for continuous parameters, in our case, two spatial dimensions. We make several standard simplifying assumptions to allow us to identify discrete states. We assume that all obstacles are convex polygons, that an agent occupies a point, and that sources and destinations of paths are points.

When obstacles are stationary, the visibility graph [Fu91] has been an important combinatorial structure in planning shortest paths. Vertices of such a graph are vertices of the obstacles as well as the given start and goal points. A line segment that connects two vertices without passing through the interior of any obstacle is an edge of the graph. A shortest path from a start point to a goal point is given as a finite sequence of edges of the visibility graph. The locations, then, to which states in our statecharts relate are the vertices on these shortest paths.

If we allow obstacles and destinations to move, we can generalize the visibility graph to an accessibility graph [Fu91]. As long as the agent can move faster than any of the obstacles and the destination, a time-minimal motion for the agent is represented as a sequence of edges of the accessibility graph. Given a velocity (speed and direction), an agent can travel a certain distance before meeting some object; this defines a *collision point* with respect to the velocity. The set of collision points around the start point with respect to a certain speed is partitioned into *collision fronts*, consisting of curves or line segments. The agent first moves to an endpoint, *e*, of one of the collision fronts computed for the start point. At *e*, the agent is at a vertex of some obstacle (which has moved while the agent moved). We now compute a set of collision fronts with *e* as start point, and the agent moves to the endpoint of one of these. This procedure is repeated until the destination point is reached. Here the locations to which states in our statecharts relate are the endpoints of collision fronts that give a shortest path from a start point to a goal point. Dependence on speed is captured by including generally narrow time intervals on the *move* actions.

In the context of planning, we view statecharts as scripts or schemas [Tu94], or possibly even previously recorded cases [Ha89], that are selected, modified, and applied as required by the case at hand. A key aspect of

applying a statechart is to identify vertices in the accessibility graph with locations used in defining the states. Planning becomes a hierarchical process, and ideally as distributed as possible. Such planning could be done in either a top-down or a bottom-up manner. In the top-down approach, the overall structure of the statechart would be selected, and one would zoom in for the detail, which ideally would be left to the subcomponents, constrained by norms inherited from supercomponents. In the bottom-up approach, we would start with statecharts for individual agents, which would make contracts to form groups and to establish norms as we zoom out to the overall structure. In either approach, the agents in a group must work out certain norms and procedures or centralize this aspect by delegating an agent to represent the collective. Either approach is tractable since we assume that we have an adequate number of statechart schemas that can be selected, modified, and applied, and we assume that there are a manageable number of norm patterns that can be applied to prune the search space. Finally, we could allow the planning to be reactive, that is, we could modify or further elaborate the plan as more information becomes available.

6. Conclusion

In this paper we have discussed how social norms can be used in plans for a society of agents. Such norms can simplify planning as well as the execution of plans. We find statecharts to be an excellent notation for representing plans for societies of agents. In particular, to express norms, we allow deontic statements (about obligations, etc.) to occur as annotations on states and components, indicating that, for example, the stated obligation applies throughout that state or to every subcomponent of that component. Also, transitions are now interpreted as normative, that is, in deontic terms. To handle norm violations, we specify certain sub-ideal, violation states, where faults are corrected. We thus also represent fault tolerance. Multi-agent motion planning in this setting involves selecting and adapting statechart schemes and identifying critical locations that can be identified with states in the statecharts. Planning can be top-down or bottom-up, but in any case it is critical to associate the agents into groups or societies and for them to accept norms.

The two most pressing topics are selection and adaptation of statechart schemes for new cases and communication among agents leading to acceptance of norms and roles. Selecting appropriate schemas requires identification of key features and ways to summarize both a statechart schema and the situation at hand so that meaningful similarities can be found with reasonable effort. Adapting statechart schemas to the situation at hand requires operators that can modify statechart structures and norms in principled ways. Agent communication actions, especially in the form of speech acts, have been studied [CL97], but generally in a context where two agents establish commitments. We must look at communication where norms encompassing entire societies of agents are accepted.

References

- [CL97] Cohen, P.R. and Levesque, H. J., "Communicative Actions for Artificial Agents", in J.M. Bradshaw (ed.), *Software Agents*. Menlo Park, CA: AAAI Press/The MIT Press, 1997, pp. 418-436.
- [Fu91] Fujimura, Kiuo, *Motion Planning in Dynamic Environments*. Tokyo: Springer-Verlag, 1991.
- [Ha89] Hammond, K. J. *Case-Based Planning: Viewing Planning as a Memory Task*. New York: Academic Press, 1989.
- [Ha87] Harel, David., "Statecharts: A Visual Formalism for Complex Systems," *Science of Computer Programming* 8, 1987. pp. 231-274.
- [La95] Lano, K., *Formal Object Oriented Development*. London: Springer-Verlag, 1995.
- [Lu97] Luu, T.V., *Enhancing Statecharts Using Deontic Logic Operators*, Technical Report, NASA Sharp Plus Program, North Carolina A&T State University, Aug., 1997.
- [MW93] Meyer, J.-J.Ch. and Wieringa, R.J., "Deontic Logic: A concise Overview, in J.-J. Ch. Meyer and R.J. Wieringa (eds.), *Deontic Logic in Computer Science: Normative System Specification*. John Wiley & Sons, 1993.
- [Tu95] Tuomela, Raimo., *The Importance of Us: A Philosophical Study of Basic Social Notions*. Stanford, CA: Stanford University Press, 1995.
- [Tu94] Turner, Roy M., *Adaptive Reasoning for Real-World Problems: A Schema-Based Approach*. Hillsdale, New Jersey: Lawrence Erlbaum Associates, 1994.