A Logical Account of Causal and Topological Maps

by

Emilio Remolina, M.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

December 2001

A Logical Account of Causal and Topological Maps

Publication No. ____________________

Emilio Remolina, Ph.D.
The University of Texas at Austin, 2001

Supervisor: Benjamin Kuipers

The Spatial Semantic Hierarchy (SSH) is a set of distinct representations for large scale space, each with its own ontology and each abstracted from the levels below it. At the control level, the agent and its environment are modeled as continuous dynamical systems whose equilibrium points are abstracted to a discrete set of distinctive states. The control laws whose execution defines trajectories linking these states are abstracted to actions, giving a discrete causal graph representation for the state space. The causal graph of states and actions is in turn abstracted to a topological network of places and paths (i.e. the topological map). Local metrical models of places and paths can be built within the framework of the control, causal and topological levels while avoiding problems of global consistency.

Most of the SSH's ideas have been traditionally described in procedural terms inspired by implementations of the SSH. This description has various problems when used to implement physical agents. First, some assumptions are not explicitly stated or are difficult to meet by current sensory technology (e.g. sensory information is not rich enough to distinguish one place from another). Second, some important SSH concepts (i.e. paths) are not properly defined or understood. Third, sometimes it is not clear what the representation states and consequently it is hard to apply it to new domains.

In this dissertation we propose a formal semantics for the SSH causal, topological and local metrical theories. Based on this semantics, we extend the SSH in the following important ways: i) we include distinctive states as objects of the theory and handle perceptual aliasing, ii) we define the models associated with the SSH causal and topological levels, iii) we extend the SSH topological theory to handle convergent and self intersecting paths as well as hierarchical maps , iv) we show how to combine causal, topological and noisy local metrical information, v) based on the previous enhancements, we define an algorithm to keep track of different topological maps consistent with the agent's experiences. This algorithm supports different exploration strategies and facilitates map disambiguation when the case arises.

Contents

Chapter 1

Dissertation Overview

The Spatial Semantic Hierarchy (SSH) [Kuipers and Byun, 1988, Kuipers and Byun, 1991, Kuipers, 1996, Kuipers, 2000] is a set of distinct representations for large scale space,[1] each with its own ontology, each with its own mathematical foundation, and each abstracted from the levels below it. The SSH is a computational theory of the cognitive map (i.e. human knowledge of large-scale space). The SSH is used in robotics to support *map building* and *navigation*. Using the SSH representation, navigation among places is not dependent on the accuracy, or even the existence, of metrical knowledge of the environment.

The SSH describes the different states of knowledge that an agent uses in order to organize its sensorimotor experiences and create a spatial representation (i.e. a map). The SSH gives an account of how continuous interaction with the world is abstracted to a discrete spatial representation. Next we describe how the different levels of the SSH (control, causal, topological and metrical) accomplish this.

At the SSH control level, the agent and its environment are modeled as continuous dynamical systems whose equilibrium points are abstracted to a discrete set of *distinctive states*. A distinctive state has an associated *view* describing the sensory input obtained at that distinctive state. The control laws whose execution defines trajectories linking these distinctive states can be abstracted to *actions*, giving a discrete causal graph representation for the state space. The causal graph of states and actions can in turn be abstracted to a topological network of *places* and *paths* (i.e. the topological map). Local metrical models, such as occupancy grids, of places and paths can then be built on the framework of the

[1]In large-scale space the structure of the environment is revealed by integrating local observations over time, rather than being perceived from a single vantage point.

topological network while avoiding problems of global consistency.[2]

Our goal is to define the SSH so that it is clear how it can be implemented across different robot platforms as well as in other domains. In principle, in order to implement the SSH on a given robot, it should be enough to define the robot's trajectory-following and hill-climbing control laws, to define views, and to show how the local metrical map is derived from sensory input. A well defined interface between the SSH and the robot's control system allows the SSH control level to identify distinctive states and actions linking them. This information is then propagated to the other levels of the SSH. Under this vision, the SSH will be a well-defined module that can be used by any robot. Unfortunately, the existing formulation of the SSH is not at this point yet.

Part of the difficulty when implementing the SSH is to extract world information through sensors. Advances in hardware and robotics, as well as experience gained from previous SSH implementations [Lee, 1996], allow us to deal with sensor unreliability and implement robust control laws. As control laws are the ones that ultimately make a robot move, it is not surprising that there has been much work on implementing the SSH control level. A side effect of this effort has been that sometimes it is not clear what is and what is not part of the SSH. For example, in [Kuipers and Byun, 1988, Lee, 1996] the robot's exploration strategy is part of the SSH control level, though the original SSH's description aims for constructing the SSH opportunistically, independent of how actions are selected and executed by the robot.

Most of the key concepts in the SSH framework have been traditionally described in procedural terms inspired by implementations of the SSH. This description has various problems when used to implement physical agents. First, some assumptions are not explicitly stated or are difficult to meet by current sensory technology (i.e. sensory information is not rich enough as to distinguish one place from another). Second, some important SSH concepts (i.e. paths) are not properly defined or understood. Third, sometimes it is not clear what the representation states and consequently it is hard to apply it to new domains.

The goal of this dissertation is to formalize the SSH causal and topological levels in order to have a clear specification of what the SSH accounts for. We define the SSH causal and topological models (maps) associated with a set of agent's experiences. These models are the spatial representation the agent creates in order to explain such experiences. How the agent explores the environment or builds such models are not part of the SSH theory.

[2]See chapter 2 for a detailed overview of the SSH.

Nevertheless, once we present the theory, we illustrate the effect of the exploration strategy as well as present algorithms to build the SSH causal and topological maps.

We start our formalization by including *distinctive states* as objects of the causal and topological theory. At the SSH control level distinctive states are the fixed points of hill climbing control laws. At the causal and topological levels distinctive states are objects representing such fixed points the agent visits at the control level. Part of the purpose of the causal and topological theories is to determine when two distinctive states refer to the same environment state.

Previous descriptions of the SSH do not include distinctive states as objects of the theory. This is the case because these descriptions assume that perceptual aliasing (i.e. different distinctive states that share the same view) does not occur. Should this hypothesis not be the case, the agent's exploration strategy has to handle any environment states disambiguation. This mix between the SSH description and the robot exploration strategy made it difficult to state what the SSH itself is about.

In this dissertation we show that by including distinctive states as first order objects of the theory it is possible to handle perceptual aliasing while formulating the spatial representation (the SSH) independently of the agent's exploration strategy. For this purpose we explicitly define the models of the causal and topological theories associated with a set of agent's experiences. Should the experiences not be complete or perceptual aliasing occur, different models of the theory may exist. In such cases, as part of the exploration strategy, the agent could choose to refute some of these models or just keep exploring the environment gathering more experiences.

One interesting aspect of spatial knowledge that becomes clear with the SSH formalization is the *non-monotonic* effect of gathering more experiences. New information could prove environment states to be different although they were previously believed equal, and viceversa. This remark applies not only to environment states but also to other SSH objects like *places* and *paths*. It is not surprising then that we use a non-monotonic logic formalism, namely Nested Abnormalities Theories (NATs) [Lifschitz, 1995], in order to state the SSH causal and topological theories.

A logical account of the SSH causal, topological and local metrical theories is given using NATs. The minimality conditions embedded in the formalization define the preferred models associated with the theories. In chapters 4 through 8 we illustrate the main properties of the new theories. In particular we show how the minimal models associated with these theories are adequate models for the spatial knowledge an agent has about its environment. We also illustrate how the different levels of the representation assume different spatial properties about both the environment and the actions performed by the agent. These spatial properties play the role of “filters” the agent applies in order to distinguish the different environment states it has visited.

As part of our formalization, we have chosen to explicitly represent when two distinctive states denote the same environment state. The predicates ceq and teq (chapters 4 and 5) denote when two distinctive states are equal given causal and topological information, respectively. ceq is the case when it is not possible to distinguish distinctive states by views (sensory input) and actions. teq is the case when distinctive states have the same view and are at the same place along the same paths. While the topological ontology (that of places and paths) is more elaborated than the one for the causal level, it is easier to build this representation than it is to distinguish environment states based only on view-action-view sequences.

At the SSH causal level we introduced a new spatial representation, that of the causal graph. A causal graph is a deterministic finite automaton (DFA) where states are ceq equivalence classes, and transitions correspond to actions. In the presence of perceptual aliasing this representation is different from the *view graph*, a non-deterministic automaton where states are views, and transitions are actions. In chapter 4 we elaborate on the properties of this representation.

The SSH topological theory is presented in chapters 5 through 8. We start with a simple topological theory that assumes “simple” paths in the environment. We then extend the theory to handle more complex paths, namely self-intersecting and convergent paths (figure 1.1). We then extend the theory with boundary regions, local metrical information and finally, abstraction regions. At each step, we illustrate how the new spatial representation allows the agent to distinguish environments not distinguishable with the previous representations. Based on the SSH causal and topological formalizations, we define an algorithm that allows the agent to keep track of different models consistent with a set of experiences.

As the ultimate goal of this work is to show that a formal specification of the SSH will facilitate its implementation on physical robots, we evaluated our methods by implementing the SSH in Vulcan, our wheelchair robot (chapter 9). As with any other representa-

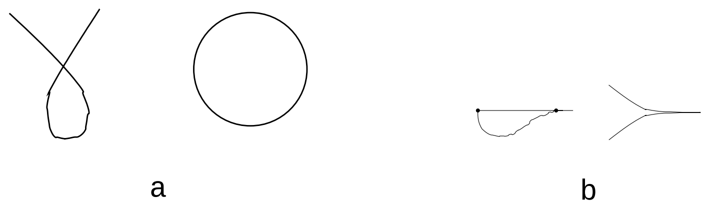


Figure 1.1: (a) Self intersecting paths. (b) Convergent paths.

tion, we should describe the processes that act upon the representation and specify how the representation supports such processes [Bickhard M., 1995, Markman, 1999]. In appendix A we illustrate how the SSH supports different architectures used for robot's navigation.

In summary, in order to bridge the gap between the SSH theory and its implementation, we propose a formal semantics of the SSH causal, topological and local metrical theories. Based on this semantics, we extend the SSH in the following important ways: i) we introduce distinctive states as first order objects of the theory and extend the theory to handle perceptual aliasing; ii) we define the models associated with the SSH causal and topological theories ; iii) we extend the theory to handle self intersecting and convergent paths; iv) we show how to combine causal, topological and noisy local metrical information; v) based on the previous enhancements, we define an algorithm to keep track of different topological maps consistent with the agent's experiences, and vi) we extend the SSH to handle hierarchical maps (i.e. maps where there is a containment relation between regions).

The rest of this document describes the ideas above in detail. Chapter 2 presents an overview of the SSH. Chapter 3 defines the SSH control level assumptions under which the causal and topological levels are built. We then go into the details of the changes we propose at the causal and topological levels. Chapters 4 and 5 present our formalization of the SSH's causal and topological levels. Chapter 6 adds boundary regions to the topological level, and chapter 8 shows how regions can be included at the SSH topological level. Chapter 7 defines the use of local metrical information in combination with causal and topological information. It also presents a method to integrate metrical information associated with paths in order to assign locations to places in a region. Chapter 9 shows our SSH implementation in Vulcan. In chapter 10 we review related research in the areas of robotics and space representation. Finally, in chapter 11 we present our conclusions.

Chapter 2

SSH Overview

2.1 The Spatial Semantic Hierarchy

The Spatial Semantic Hierarchy (SSH) [Kuipers and Byun, 1988, Kuipers and Byun, 1991, Kuipers *et al.*, 1993, Kuipers, 1996, Kuipers, 2000][1] is an *ontological hierarchy* of representations for knowledge of large-scale space. An ontological hierarchy shows how multiple representations for the same kind of knowledge can coexists. Each level of the hierarchy has its own *ontology* (the set of objects and relations it uses for describing the world) and its own set of inference and problem-solving methods. The objects, relations, and assumptions required by each level are provided by those below it.

The SSH abstracts the structure of an agent's spatial knowledge in a way that is relatively independent of its sensorimotor apparatus and the environment within which it moves. Next we describe the different SSH levels.

- The *sensorimotor level* of the agent provides continuous sensors and effectors, but not direct access to the global structure of the environment, or the robot's position or orientation within it.

- At the *control level* of the hierarchy, the ontology is an egocentric sensorimotor one, without knowledge of fixed objects or places in an external environment. A *distinctive state* is defined as the local maximum found by a hill-climbing control strategy, climbing the gradient of a selected feature, or *distinctiveness measure*. Trajectory-following control laws take the robot from one distinctive state to the neighborhood of the next, where hill-climbing can find a local maximum, reducing position error and preventing its accumulation.

[1]This presentation follows [Kuipers, 1996].

- The ontology at the SSH *causal level* consists of views, distinctive states, actions and schemas. A *view* is a description of the sensory input obtained at a locally distinctive state. An *action* denotes a sequence of one or more control laws which can be initiated at a locally distinctive state, and terminates after a hill climbing control law with the robot at another distinctive state. A *schema* is a tuple $\langle\, (V, dp), A, (V', dq)\, \rangle$ representing the (temporally extended) event in which the robot takes a particular action A, starting with view V at the distinctive state dp, and terminating with view V' at distinctive state dq. The spatial representation posits the minimal set of distinctive states consistent with the set of schemas.

- At the *topological level* of the hierarchy, the ontology consists of *places*, *paths* and *regions*, with connectivity and containment relations. The spatial representation posits the minimum set of paths and places consistent with the set of causal schemas.[2] At the SSH topological level, action symbols are categorized in two classes: *Turn* and *Travel*. A place corresponds to a set of distinctive states linked by turn actions. A path is a structure that includes an ordered sequence of places connected by travel actions without turns. Paths are used in the cognitive map to describe linear geographical structures such as streets. Places and paths define a topological network which can be used to guide exploration of new environments and to solve new route-finding problems.[3] Using the network representation, navigation among distinctive states is not dependent on the accuracy, or even the existence, of metrical knowledge of the environment.

- At the *metrical level* of the hierarchy, the ontology for places, paths, and sensory features is extended to include metrical properties such as distance, direction, shape, etc. Geometrical features are extracted from sensory input, and represented as annotations on the places and paths of the topological network.

Two fundamental ontological distinctions are embedded in the SSH. First, the continuous world of the sensorimotor and control levels is abstracted to the discrete symbolic representation at the causal and topological levels, to which the metrical level adds continuous properties. Second, the egocentric world of the sensorimotor, control, and causal level is abstracted to the world-centered ontologies of the topological and metrical levels.

[2]In order to state these minimality conditions, the causal and topological levels are formalized as circumscriptive theories (see chapter 5).

[3]Notice that although the topological map has a graph like structure, a path in the graph theory sense is not necessarily a SSH topological path.

2.2 Creating Schemas

As the agent navigates its environment, a set of schemas summarizing its experiences is created. This set of schemas is the only source of information the agent has to create a spatial representation of its environment. Next we describe how a set of schemas is created as the agent navigates through its environment.

At the SSH control level, exploration is performed by alternating execution between two types of continuous control strategies,[4] *trajectory-following* and *hill-climbing* . These two types of control strategy differ in their roles: a *hill-climbing* control strategy is for climbing towards a local maximum of a distinctiveness measure and thus a position of some distinctive state; a *trajectory-following* control strategy is for moving from the neighborhood of one distinctive state to the neighborhood of another. The actual motion from one distinctive state to the neighborhood of another may be the result of the execution of a sequence of more than one control strategy (see example below). Let $cl = cl_1, \ldots, cl_m$ be the sequence of control strategies executed at the control level to take the agent from distinctive state ds with view V to distinctive state ds' with view V'. Then, the schema $\langle (V, ds), A, (V', ds') \rangle$ is created at the SSH causal level, where A is an *action* symbol used whenever the sequence cl is executed.[5] This way, the experiences of the robot within its environment can be described by an alternating sequence of views and actions

$$(V_1, ds_1)\, A_1\, (V_2, ds_2)\, \ldots A_{n-1}\, (V_n, ds_n)$$

which is summarized at the causal level by the set of schemas S,

$$S = \{ \langle\, (V_i, ds_i), A_i, (V_{i+1}, ds_{i+1})\, \rangle \, : \, i = 1, \ldots, n-1 \}$$

Example 1

Consider the environment in figure 2.1. In order to go from distinctive state $ds1$ to distinctive state $ds2$, the agent executes the sequence of control strategies $\langle$`get-into-corridor, follow-middle-line, hc-T-intersection`$\rangle$ where *get-into-corridor* is a trajectory-following control strategy that moves the agent from $ds1$ to `a`, `follow-middle-line` is a trajectory-following strategy that takes the agent from `a` to `b`, and `hc-T-intersection` is a hill-climbing control strategy that takes the agent from `b` to the distinctive state $ds2$. Environment states `a` and `b` are not distinctive states. At the distinctive state $ds2$ the agent is facing the wall ahead and it is equidistant from this wall and the intersection corners.

[4]These strategies correspond to continuous control laws [Kuo, 1987].

[5]See section 3.1.3, page 14.

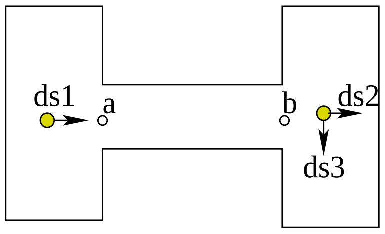


Figure 2.1: A sequence of control strategies , ⟨*get-into-corridor, follow-middle-line, hc-T-intersection*⟩, takes the agent from distinctive state $ds1$ to distinctive state $ds2$. At the causal level, this continuous motion is represented by the schema $\langle\, (V, ds1), A, (V', ds2)\, \rangle$, where the action symbol A represents the sequence ⟨*get-into-corridor, follow-middle-line, hc-T-intersection*⟩, V and V' are the views at $ds1$ and $ds2$ respectively.

Distinctive state $ds3$ is at the same physical location as $ds2$ but with a different orientation. When the robot is at ds3, it is facing the open space (corridor) at the right of $ds2$. In order to go from distinctive state $ds2$ to distinctive state $ds3$, the agent executes the sequence of control strategies ⟨`face-space-on-right,align-with-corridor`⟩.

At the causal level, the schemas $\langle (V1, ds1), A1, (V2, ds2) \rangle$ and $\langle (V2, ds2), A2, (V3, ds3) \rangle$ are created, where $A1$ represents the sequence ⟨`get-into-corridor, follow-middle-line, hc-T-intersection`⟩ and $A2$ represents the sequence ⟨`face-space-on-right,align-with-corridor`⟩. At the topological level, $A1$ is a `Travel` action while $A2$ is a `Turn` action.

{*end of example*}

2.3 The SSH topological level: regions

The SSH topological level includes *regions*, *boundaries* and *containment* relations. A *boundary* is a sequence of one or more directed paths. A *region* is a set of places. A *boundary region* is the set of places defined to be on one side of a boundary. A path has associated two boundary regions, its right and left sides.

Boundary region information can be extracted from sensorimotor experiences ([Kuipers and Levitt, 1988] suggests how to do this though it does not defines the general mechanism for doing so). Specifying where a place lies with respect to a dividing boundary provides partial knowledge about its position. This knowledge is particularly easy to acquire, easy to combine with other similar pieces of knowledge, and easy to apply to route-finding problems. Partial states of knowledge are possible since the agent does not know the relative position of every place with respect to every boundary. Once a sufficient number of boundary relations have been accumulated, they provide a useful topological route-finding heuristic. For example, to find a route from A to B, if there is a path such that A is on its right and B is on its left, look for routes from A to that path and from the path to B. In [Remolina and Kuipers, 1998b] we have given a formal account of how boundary relations are established as the agent navigates a known environment. In chapter 6 we show how boundary regions are used while exploring an unknown environment.

2.4 Physical Implementation of the SSH

An implementation of the SSH in a physical robot (Spot) was carried out by W.Y. Lee [Lee, 1996]. Most of this work focused on the SSH control level since the other levels can be constructed as in the TOUR model [Kuipers, 1978].[6] Next we review the key ideas in the implementation.

At the SSH control level, the robot has only an egocentric view of its surroundings with no concept of external objects. Sensory regularities are the way to identify external objects. Discontinuities in sensory readings mark the presence of distinctive neighborhoods. A SSH robot identify and responds only to sensorimotor regularities at the control level.[7]

The type of possible distinctive neighborhoods the robot can be at is given a priory. Each of these neighborhoods types has associated a hill climbing strategy as well as a preselected order in which distinctive states are visited. Local frames of reference (LFOR) are created when visiting a neighborhood. This coordinate system allows the creation of views as well as the tracking of corners otherwise not possible from sonar readings. Since the sensory information associated with a neighborhood is not at once available to the robot, the robot has to explore the neighborhood in order to identify the neighborhood's type. This exploration procedure is described by a finite state automaton. Final states in this automaton identify the type of neighborhood. Transitions in the automaton describe how information of the neighborhood should be collected and used to discriminate the neighborhood.

In order to choose a hill climbing strategy, the following elements are considered:

[6]Notice that the TOUR model does not account for perceptual aliasing, and it assumes that a unique topological map is consistent with the agent's experiences. Any possible ambiguity has to be solved before giving the schemas to the TOUR algorithm.

[7]For example, a frontal object is represented internally as a regularity of the changes in frontal sonar measurement over time [Lee, 1996].

the trajectory-following control strategy that brought Spot into the neighborhood, the event that signal the neighborhood (corner or front-blocked), and an analysis of the immediate surroundings in terms of qualitative positions of frontal objects, if any. The hill climbing strategies work by applying a trajectory-following control strategy that constrains two degrees of freedom, and then climbing a distinctive measure in the remaining one degree of freedom until its local maximum is reached. In general the hill climbing strategy is a two step procedure: first, a distinctive location must be found, and then distinctive orientations at that location are determined. A distinctive state is a pair *(location, orientation)*. Special algorithms to find distinctive orientations were developed. These algorithms deal with error in sonar readings produced by specularity.

Chapter 3

Control Level

At the SSH control level, the agent and its environment are modeled as continuous dynamical systems whose equilibrium points are abstracted to a discrete set of *distinctive states*. The agent "moves" from one distinctive state to another, by using a combination of *trajectory following* and *hill climbing* control laws. The execution of these control laws is abstracted at the SSH causal level as a set of causal schemas (section 2.2, page 8) from which causal, topological and metrical representations of space are created (chapters 4 to 8). Our main hypothesis to build these spatial representations is that the execution of control laws is *deterministic* in the following sense: executing a sequence of control laws at a particular environment state leaves the agent "at the same" resulting distinctive state. In this chapter we state this property of the SSH control level (the SSH *closure property*).

In general it is difficult to prove that a given set of control laws and a particular environment satisfy the SSH closure property. A particular case where the closure property can be proved is for a "Voronoi robot", a robot that moves along the Voronoi diagram associated with the environment: the set of points that are equidistant from two nearby objects [Aurenhammer, 1991, Choset and Nagatani, 2001] . In section 3.2 we elaborate in the properties and limitations of Voronoi robots.

Hill climbing control laws are chosen based on sensory input and (maybe) the trajectory following that brought the robot to the current environment state. It may be the case that different hill climbing control laws bringing the robot to different distinctive states could be chosen for execution. We would rule out this possibility by requiring distinctive states to be *well separated* (section 3.1.2).

The closure property on the set of control laws and the well separation among distinctive states guarantee that actions at the causal level are deterministic. *For the purpose of this dissertation, this is all we need to know about the control level.* Next we briefly elaborate on these SSH control level properties. The reader is referred to [Lee, 1996, Kuipers, 2000] for more detailed information.

3.1 SSH control assumptions.

[1]Exploration of an unknown environment takes place by selecting a control law based on sensory information available about the local neighborhood. Typically, we expect behavior to be an alternation between *hill-climbing control* laws, which bring the agent to a locally-distinctive state from any state within the local neighborhood, and *trajectory-following* control laws, which bring the agent from one distinctive state to the neighborhood of the next.

3.1.1 The SSH control closure property

The navigation strategy of alternating trajectory-following and hill-climbing control laws presumes that the following criteria are satisfied. We call these the *closure criteria* on the set of control laws.

1. After a hill-climbing control law is executed and terminates at a distinctive state, at least one trajectory-following control law is available for selection. This ensures that there is a choice of action from the current distinctive state: there are no dead end.

2. After a trajectory-following control law is executed and reaches its termination state, at least one hill-climbing control law is available for selection. This ensures that each trajectory terminates at a distinctive state.

3.1.2 Well separated dstate

The closure property does not rule out the possibility that more than one hill-climbing may be available for selection. For the purpose of having deterministic actions at the causal level, we require that

> the basins of attraction of distinctive states are well separated.

This separation property ensure that at most one hill-climbing is available for selection after a trajectory following control law is performed. Notice that the basins of attraction of

[1]In this section we follow [Kuipers, 2000].

distinctive states could intersect, but in those common environment states the robot has a consistent “clear choice” of which hill climbing control law to perform (see figure 3.1).

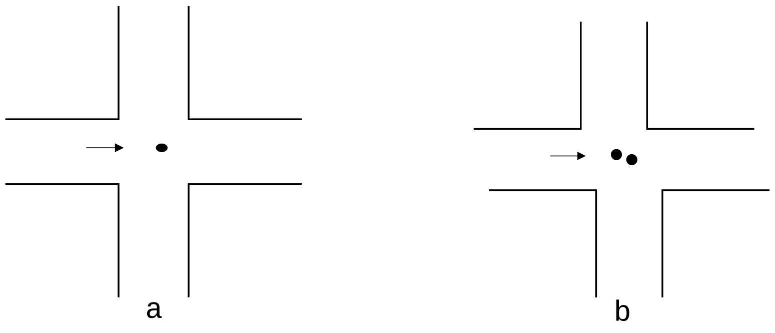


Figure 3.1: Consider a robot coming into an intersection as indicated by the arrow in the figure. Suppose a robot hill climbs to localize itself equidistant from at least three nearby objects. (a) In a “perfect” intersection, the agent will hill climb to the center of the intersection. (b) A small perturbation on the intersection of a, will cause the robot to have at least two different distinctive states it could reach (indicated by dots in the figure). The SSH well-separated distinctive states property requires the agent to have a clear selection criteria so that when entering the intersection the agent always hill climbs to the same distinctive state.

3.1.3 From control laws to actions

Sequences of control laws are given an *action* name at the SSH causal level. In order to decide whether two of such sequences have the same name, we do not consider the last control law of the sequence (which is required to be a hill climbing control law). Formally,

$$action(cl_1, \ldots, cl_n) = action(cl'_1, \ldots, cl'_m) \; iff \; n = m \wedge \forall i < n \; cl_i = cl'_i$$

where $action(cl)$ denotes the action name associated with the sequence of control laws cl.

3.2 Voronoi robots

A Voronoi robot moves along the Voronoi diagram associated with the environment: the set of points that are equidistant from two nearby objects [Choset and Nagatani, 2001]. Voronoi robots are relevant to the SSH control level because they have a well developed mathematics that allows one to characterize and prove properties about the control level. Moreover, they are well understood and different algorithms exist to implement them [Choset *et al.*, 1997, Choset and Nagatani, 2001]. However, they have two well known limitations. First, small perturbations in the environment can greatly change the Voronoi diagram (for instance, consider the example in figure 3.1). Second, they assume that the robot can perceive at least two nearby objects. This is not the case if the agent only has weak sensors or if objects are outside the range of the sensors. In this case, control laws that keep the robot equidistant from a reference wall are used [Kuipers and Byun, 1988, Lee, 1996].

For the purpose of this dissertation all we need to assume about the SSH control level is that its behaviors can be abstracted to deterministic actions at the causal level. This is guarantee by satisfying the closure and well-separation properties. Certainly, actions could fail and so the agent could get lost. We handle these failures during navigation (when using an already built map) and assume they do not occur during map building.

Chapter 4

Causal Level

The agent's experiences in the environment are summarized at the SSH causal level by *schemas*. A *schema* represents an agent's particular action execution in the environment. An action execution takes the agent from one *distinctive state* to another. Sensory information at distinctive states is represented by *views*.

We can think of a schema as a tuple

$$\langle (v, ds), a, (v', ds') \rangle$$

representing the event in which the agent starting at distinctive state ds (whose view is v), executed action a, terminating at distinctive state ds' (whose view is v'). It is also possible to have an incomplete schema where the resulting distinctive state, ds', is "missing". These incomplete schemas allow the SSH causal level to account for common states of incomplete knowledge like "I could take you there, but I can't tell you how" [Kuipers, 2000]. In section 4.1.4 we define how to represent schemas and the information associated with them. We also formally define (see section 4.1.5) a variety of useful "tuple notations", including $\langle (v, ds), a, (v', ds') \rangle$, which we will use hereafter.

Schemas can be used to direct the agent behavior in the environment, to distinguish distinctive states, or as a basis for more elaborated spatial representations. In the first case, for example, if the agent's current view is v and it has experienced a schema $\langle (v, ds), a, (v', ds') \rangle$, it could execute a and expect to observe v' [Kuipers, 2000]. In the second case, by considering view-action-view sequences, the agent can distinguish distinctive states that share the same view (see section 4.2.3). Finally, in chapter 5 we show how by adding some spatial interpretation to the actions executed by the agent, we can build a spatial representation from a set of schemas (the SSH topological map).

This chapter is organized as follows: section 4.1 presents the causal level ontology. We then define how causal information can be used to distinguish distinctive states (section 4.2). We define two types of "spatial" representations that can be directly derived from a set of schemas: the SSH view graph (section 4.2.2) and the SSH causal graph (section 4.3). Finally, in section 4.4 we present a logic program to calculate the models of the SSH causal level theory.

4.1 Causal level Ontology

We use a first order sorted language in order to describe the SSH causal level. The sorts of such language include *views*, *actions*, *distinctive states*, *schemas* and *routines*.[1] Next we present the predicate symbols and axioms associated with each of these sorts.

4.1.1 Views

A *view* represents a set of sensory inputs. While it is possible to associate a view with any environment state, only views associated with sensory input at distinctive states are considered by the SSH. Moreover, at the SSH causal level only the name of the view matters. Different view names represent different sets of sensory input (see axiom 4.15). The internal structure used by the agent to describe a sensory input is not considered at the SSH causal level.

4.1.2 Actions

An *action* denotes a sequence of one or more control laws. As for views, at the SSH causal level only the name of the action matters. Different action names represent different sequences of control laws (see axiom 4.14).

We assume that an action has a type, either *travel* or *turn*, associated with it.[2] We use the predicate

$$Action\_type(a, type) \quad ,$$

[1]New sorts will be added when we present the SSH topological and metrical levels (chapters 5 and 7).

[2]The type of an action will be important at the SSH topological level (chapter 5). For completeness of the presentation we introduce this concept here.

to represent the fact that the type of action *a* is *type*. The type of an action is unique:[3] [4]

$$\exists! type \;\; Action\_type(a, type) \;\; . \tag{4.1}$$

The constant symbols *turn* and *travel* define completely the sort of *action_types*. Formally,[5]

$$\begin{aligned} & turn \neq travel \;, \\ & \forall atype \;\; \{atype = turn \vee atype = travel\} \;\; . \end{aligned} \tag{4.2}$$

Turn Right, Turn left, Turn Around

Turn actions have associated a qualitative description. This new sort of qualitative descriptions is completely defined by the different constant symbols *turnLeft, turnRight* and *turnAround*. Formally,[6]

$$UNA(turnLeft, turnRight, turnAround) \;, \tag{4.3}$$

$$\forall desc \;\; \{desc = turnLeft \vee desc = turnRight \vee desc = turnAround\} \;\; . \tag{4.4}$$

We use the predicate

$$Turn\_desc(a, desc)$$

to indicate that *desc* is the qualitative description of *action a*. The description associated with an action is unique. Moreover, only turn actions have associated a qualitative description:

$$Turn\_desc(a, desc) \rightarrow Action\_type(a, turn) \;, \tag{4.5}$$

$$Action\_type(a, turn) \rightarrow \exists! desc \; Turn\_desc(a, desc) \;\; . \tag{4.6}$$

Axiom 4.5 says that only *turn* actions have associated a qualitative description. Axiom 4.6 states that each turn action has associated a unique qualitative description.[7]

[3]Throughout this paper we assume that formulas are universally quantified.

[4]The formula $\exists! v\, P(v)$ means *"there exists a unique v s.t.* $P(v)$*"*. Formally, $\exists v \forall x\, [P(x) \equiv x = v]$.

[5]While we assume a first order sorted logic, such a logic does not have "subsorts". It will be nice to say: the sort of actions has two subsorts, turn actions and travel actions. In order to have the "subsorts" of turn and travel actions, we will have to explicitly include the predicates "turn" and "travel" (instead of the constant symbols $turn$ and $travel$), and require that

$$\forall a \;\; turn(a) \equiv \neg travel(a) \;\; .$$

We have chosen not to explicitly represent "subsorts of actions" but rather talk about the "type of the action".

[6]The notation $UNA(t_1, \ldots, t_n)$ represents the uniqueness of names axioms for the grounded terms $t_1, \ldots, t_n$. These axioms simply require that $t_i \neq t_j$ for $i \neq j$.

[7]We could think of the turn action's qualitative description as defining the "turn actions subsorts" turnRight, turnLeft and TurnAround (see footnote 5).

4.1.3 Distinctive States

At the SSH control level a distinctive state is defined as the local maximum found by a hill-climbing control strategy, climbing the gradient of a selected feature, or distinctiveness measure. At the SSH causal level, names are given to these distinctive states. The agent associates distinctive state names with the environment state it is at after performing a hill climbing control law. It is possible for the agent to associate different distinctive state names with the same environment state. This is the case since the agent might not know at which of several environment states it is currently located.

A distinctive state has an associated view. We use the predicate

$$View(ds, v) \ ,$$

to represent the fact that *v* is a *view* associated with *distinctive state* ds. We assume that a distinctive state has a unique view,

$$\exists! v \, View(ds, v) \ . \tag{4.7}$$

However, we do **not** assume that views uniquely determine distinctive states (i.e. $View(ds, v) \wedge View(ds', v) \not\rightarrow ds = ds'$). This is the case since the sensory capabilities of an agent may not be sufficient to distinguish distinctive states.

4.1.4 Schemas

A schema represents a particular action execution of the agent in the environment. An action execution is characterized in terms of the distinctive states the agent was at before and after the action was performed. We use the following predicates to represent information associated with a schema:

- *action(s,a)*: *action* a is the action associated with *schema s*.
- *context(s,ds)* : ds is the starting *distinctive state* associated with the action execution represented by *schema* s.
- *result(s,ds)* : ds is the ending *distinctive state* associated with the action execution represented by *schema* s.

While we require a unique context and action associated with a schema, the result of a schema is optional (but unique if it exists):

$$\exists! a \ action(s, a) \ , \tag{4.8}$$

$$\exists! ds \ context(s, ds) \ , \tag{4.9}$$

$$result(s, ds) \wedge result(s, ds') \rightarrow ds = ds' \ . \tag{4.10}$$

Most often we are interested in *complete* schemas: those for whom the resulting distinctive state exists. We use the predicate $CS(s, ds, a, ds')$ defined as

$$CS(s, ds, a, ds') \equiv_{def} context(s, ds) \wedge action(s, a) \wedge result(s, ds') \tag{4.11}$$

to express the fact that schema *s* represents an execution of action *a* which took the agent from *distinctive state* ds to distinctive state ds'.[8]

An action execution also has metrical information associated with it. This metrical information represents an estimate of, for example, the distance or the angle between the distinctive states associated with the action execution. We defer the study of metrical information associated with schemas until chapter 7.

4.1.5 Schema notation

While schemas are explicit objects of our theory, most of the time it is convenient to leave them implicit. We introduce the following convenient notation:

$$\langle ds, a, ds' \rangle \quad \equiv_{def} \quad \exists s \; CS(s, ds, a, ds')$$

$$\langle v, a, v' \rangle \quad \equiv_{def} \quad \exists s, ds, ds' \;\; \{CS(s, ds, a, ds') \wedge View(ds, v) \wedge View(ds', v')\}$$

$$\langle (v, ds), a, (v', ds') \rangle \quad \equiv_{def} \quad \exists s \;\; \{CS(s, ds, a, ds') \wedge View(ds, v) \wedge View(ds', v')\}$$

$$\langle ds, type, ds' \rangle \quad \equiv_{def} \quad \exists s, a \;\; \{CS(s, ds, a, ds') \wedge Action\_type(a, type)\}$$

$$\langle ds, desc, ds' \rangle \quad \equiv_{def} \quad \exists s, a \;\; \{CS(s, ds, a, ds') \wedge Turn\_desc(a, desc)\}$$

Notice that we have "overloaded" the bracket notation depending on the type of its arguments.

4.1.6 Routines

A *routine* is a set of schemas indexed by views. We use the predicate

$$Routine(r, v, s)$$

[8]CS stands for Causal Schema.

to indicate that in *routine r, schema s* is indexed by view *v*. A view can index multiple schemas in a routine. That a view indexes a schema means that the context of the schema should have associated such view:

$$Routine(r, v, s) \wedge context(s, ds) \rightarrow View(ds, v)$$

Routines model routes where particular actions are taken when the agent observes a given view. Routines are used to model "situated action" where the agent chooses its next action to execute by choosing a schema associated with the current view. It is possible that the current view indexes more than one schema in which case either a non-deterministic choice is made or if the agent is paying enough attention to identify the distinctive state associated with the view, then that will allow the schema to be selected deterministically. Notice that at a distinctive state different actions can be performed, and consequently the agent may have different schemas associated with a distinctive state. However, when schemas are indexed in a routine, only one schema per distinctive state is indexed:

$$Routine(r, v, s) \wedge Routine(r, v, s') \wedge context(s, ds) \wedge context(s', ds) \rightarrow s = s'$$

While routines allow the SSH to explain different phenomena associated with human navigation abilities (e.g. leaving home to buy groceries on Saturday morning and ending up at work), there is not a more complete theory about routines than the one presented here. We have left as a future work to formally describe a navigation model using routines. When complemented with places (see next chapter), routines account for learned plans the agent could perform even when such plans are partially specified.

4.2 SSH Causal theory

The agent's experiences in the environment are described in terms of *CS*, *View*, *Action_type* and *Turn_desc* atomic formulae. Hereafter we use *E* to denote a particular agent's experience formulae. Given *E* we want to specify how the agent can distinguish different distinctive states. Informally, distinctive states ds and ds' are distinguishable at the SSH causal level, if either they have different views or there exists a sequence of actions "connecting" these distinctive states to corresponding distinguishable distinctive states (section 4.2.3).

4.2.1 The E formulae.

As mentioned above, the agent's experiences in the environment, *E*, are described in terms of *CS*, *View*, *Action_type* and *Turn_desc* atomic formulae. Associated with *E* we have the following set of constant symbols occurring in *E*:

- S(E) : the set of *schema* constant symbols occurring in *E*.
- DS(E) : the set of *distinctive states* constant symbols occurring in *E*.
- V(E) : the set of *view* constant symbols occurring in *E*.
- A(E) : the set of *action* constant symbols occurring in *E*.

We require the *uniqueness of names* assumption for the different constant symbols occurring in *E*:

$$UNA[s_1, \ldots, s_k] \qquad s_i \in S(E) \tag{4.12}$$
$$UNA[ds_1, \ldots, ds_l] \qquad ds_i \in DS(E) \tag{4.13}$$
$$UNA[a_1, \ldots, a_n] \qquad a_i \in A(E) \tag{4.14}$$
$$UNA[v_1, \ldots, v_m] \qquad v_i \in V(E) \tag{4.15}$$

The uniqueness of names axioms above are not only required from a logical point of view, but make sense from the knowledge representation point of view. Each of the agent schemas represents a different experience and the agent names them with a different schema constant symbol (axiom 4.12).[9] Similar remark applies for the names of actions (axiom 4.14). We assume that different view symbols represent different sensory input (axiom 4.15). This is the case since the agent decides what view to associate with a sensory input. As for the distinctive state symbols in E, *different distinctive state constant symbols might represent the same environment state.* Part of the objective of the SSH causal and topological theories is to conclude which distinctive state symbols are to be interpreted as the same environment states. Nevertheless, we assume that different distinctive state symbols in E are interpreted by different states (axiom 4.13) and we use the predicate ceq to indicate whether two distinctive states represent the same environment state.[10] In this case we said that the distinctive states are *causally* indistinguishable (see section 4.2.3).[11]

We require the *domain closure* assumption for the sort of *views*, *distinctive states*, *schemas* and *actions*. These domain axioms state that the only views, distinctive states, actions or schemas that exist are those explicitly named by the symbol constants occurring in *E*. These axioms prevent models of the SSH from including objects different from those

[9]From an implementation point of view, each schema is represented by a unique frame (or data structure) in the database.

[10]*ceq* stands for Causally Equal.

[11]The SSH causal level might not be enough to distinguish distinctive states experienced at different environment states (see Example 4, page 29).

experienced (named) by the agent.

$$\forall s \bigvee_{s_i \in S(E)} s = s_i \tag{4.16}$$

$$\forall ds \bigvee_{ds_i \in DS(E)} ds = ds_i \tag{4.17}$$

$$\forall a \bigvee_{a_i \in A(E)} a = a_i \tag{4.18}$$

$$\forall v \bigvee_{v_i \in V(E)} v = v_i \tag{4.19}$$

Finally, the type of actions as well as the qualitative description of turn actions have to be specified as part of the formulae E:[12]

$$Action\_type(a, type) \equiv \bigvee_{Action\_type(a_i, type_i) \in E} [a = a_i \wedge type = type_i] \tag{4.20}$$

$$Turn\_desc(a, desc) \equiv \bigvee_{Turn\_desc(a_i, desc_i) \in E} [a = a_i \wedge desc = desc_i] \tag{4.21}$$

Definition 1

Given a set E of *CS*, *View*, *Action_type* and *Turn_type* atomic formulae,

$$COMPLETION(E)$$

denotes the union of *E* with Axioms 4.12 - 4.21.
{*end of definition*}

The next example illustrates the concepts and axioms defined in this section.

Example 2

Consider the set of experiences *E* gathered by the agent while navigating the environment in figure 4.1. The agent moves among intersections by performing a midline control law, which at the SSH causal level becomes action *ml*. The sensory input at the different intersections is very similar, and the agent associates the view $v+$[13] with the different distinctive

[12]While we give an explicit formula for the completion of $Action\_type$ and $Turn\_desc$, we could have written these axioms as $CIRC(E; Action\_type)$, $CIRC(E; Turn\_desc)$ expressing the fact that the domains (extents) of $Action\_type$ and $Turn\_desc$ should be the smallest ones given E. Proposition 2 in [Lifschitz, 1994] shows that these two formulations are identical. As usual, "completion axioms" rule out models where the agent uses information other that the explicitly stated in E. See appendix C, page 219.

[13]As with any other symbol name, the view name is arbitrary. The + in the view name is used to indicate that the view corresponds to a four corridor intersection. Later we use the symbol $\sqsupset$ to indicate that the view corresponds to an end of corridor.

states it found (i.e. *a*, *b* and *c*).

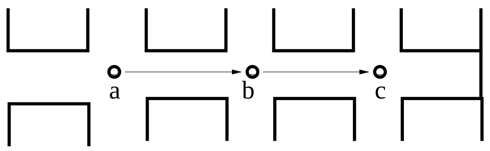


Figure 4.1: The agent moves among corridor intersections that have the same view $v+$. a, b and c are the distinctive states where this view is observed at.

The elements of *E* are as follows:

$$
\begin{array}{l}
Action\_type(ml, travel)\ , \\
CS(s1, a, ml, b)\ ,\ CS(s2, b, ml, c)\ , \\
View(a, v+)\ ,\ View(b, v+)\ ,\ View(c, v+)\ .
\end{array}
$$

The uniqueness of names axioms associated with *E* are:

$$
\begin{array}{l}
s1 \neq s2\ , \\
a \neq b \wedge a \neq c \wedge b \neq c\ .
\end{array}
$$

The domain closure axioms associated with *E* are:

$$
\begin{array}{l}
\forall s\, \{s = s1 \vee s = s2\}\ , \\
\forall ds\, \{ds = a \vee ds = b \vee ds = c\}\ , \\
\forall a\ a = ml\ , \\
\forall v\ v = v+\ .
\end{array}
$$

Finally, we also have the axioms

$$
\begin{array}{l}
\forall a, type\ Action\_type(a, type) \equiv [a = ml \wedge type = travel]\ , \\
\forall a, desc\ Turn\_desc(a, desc) \equiv false\ .
\end{array}
$$

{*end of example*}

4.2.2 The SSH view graph

Schemas can be used to define further representations of the agent's experiences in the environment. Two of such representations are the *view graph* and the *SSH causal graph*. Next we present the view graph and defer the presentation of the SSH causal graph to section 4.3.

The *SSH view graph* associated with a set of experiences E is the labeled graph $\langle Nodes, Edges, Labels \rangle$ such that:

- Nodes = V(E).
- Labels = A(E).
- Edges = $\{(v,a,v') \ : \ COMPLETION(E) \models \langle v,a,v' \rangle \ \}$.

Example 3

The view graph associated with the set of experiences in example 2 is depicted in figure 4.2. $\{$*end of example*$\}$

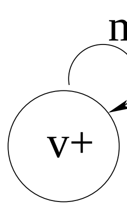


Figure 4.2: View graph associated with the experiences in example 2.

As the example above suggests, the view graph is not very informative when the same view occurs at different environment states. The agent has to use information other than the views alone in order to distinguish different environment states (for example, see chapter 5). However, should the agent have enough sensory capabilities as to distinguish distinctive states by their views, then the view graph becomes a powerful spatial representation for reliable navigation. Work in [Schölkopf and Mallot, 1995, Franz *et al.*, 1998, Mallot and Gillner, 2000, Steck and Mallot, 2000] shows how the view graph is consistent with human navigation abilities.

In the next section we illustrate how using actions and distinctive states the agent can distinguish environment states. In the next chapter we show how topological information can be used to further distinguish environment states.

4.2.3 CT(E)

Given a set of experiences *E*, the SSH causal theory *CT(E)* defines when two distinctive states are indistinguishable at the SSH causal level. We use the predicate

$$ceq(ds, ds')$$

to denote the fact that *distinctive states* ds and ds' are *causally* indistinguishable.[14] Informally, $ceq(ds, ds')$ is the case whenever distinctive states ds and ds' are indistinguishable by the actions and views in E. We will assume that actions are *deterministic*. Whenever the agent performs an action at a given distinctive state, it always ends up at the same distinctive state:

$$\langle ds, a, ds' \rangle \wedge \langle ds, a, ds'' \rangle \rightarrow ds' = ds'' \ . \tag{4.22}$$

The SSH causal theory associated with a set of experiences *E*, *CT(E)*, is the following nested abnormality theory (NATs) [Lifschitz, 1995] (see appendix C, page 219):[15]

$$\begin{aligned} & CT(E) \ = \\ & \quad COMPLETION(E)\ , \\ & \quad Axioms\ 4.1 - 4.11\ , \\ & \quad \langle ds, a, ds' \rangle \wedge \langle ds, a, ds'' \rangle \rightarrow ds' = ds'', \qquad (Axiom\ 4.22) \\ & \quad CEQ\_block \end{aligned} \tag{4.23}$$

where **CEQ_block** is:

$$\begin{aligned} & CEQ\_block \ = \\ & \quad \{\ max\ \ ceq : \\ & \quad\ ceq(ds_1, ds_2) \rightarrow ceq(ds_2, ds_1), \\ & \quad\ ceq(ds_1, ds_2) \wedge ceq(ds_2, ds_3) \rightarrow ceq(ds_1, ds_3), \\ & \\ & \quad\ ceq(ds_1, ds_2) \rightarrow View(ds_1, v) \equiv View(ds_2, v), && (4.24) \\ & \quad\ ceq(ds_1, ds_2) \wedge \langle ds_1, a, ds_1' \rangle \wedge \langle ds_2, a, ds_2' \rangle \rightarrow ceq(ds_1', ds_2') && (4.25) \\ & \quad \} \end{aligned}$$

Next we discuss the axioms defining ceq. First at all, the predicate ceq is an equivalence relation on the sort of distinctive states.

Theorem 1 *The predicate ceq is an equivalence relation.*

[14]In chapter 5 we define when distinctive states are topologically indistinguishable.

[15]Throughout this paper we assume that formulas are universally quantified.

Proof. See appendix E (page 229).[16]

Axiom 4.24 states that indistinguishable distinctive states have the same view. Finally, axiom 4.25 states that if distinctive states ds and ds' are indistinguishable, then it should be the case that if action *a* has been performed for both ds and ds', the resulting distinctive states should be indistinguishable. This axiom captures the following intuition: if ds and ds' are two indistinguishable distinctive states, any sequence of actions executed at ds and ds' will render the same sequence of views. Indeed, axioms 4.24 and 4.25 allow us to prove the following useful lemma:

Lemma 1 *Let A denote a sequence of action symbols. Let $A(ds)$ denote the distinctive state symbol resulting from excuting the sequence A starting at distinctive state ds, or $\perp$ if A is not defined for ds.*[17] *Then,*

$$\begin{aligned} &ceq(ds_1, ds_2) \wedge A(ds_1) \neq \perp \wedge A(ds_2) \neq \perp \\ &\quad \rightarrow View(A(ds_1), v) \equiv View(A(ds_2), v) \ . \end{aligned}$$

There is a special case when *ceq* is an equivalence relation without writing the axioms stating so. This happens when the result of every action at every distinctive state is known. In this case, we say that the set of experiences is *complete*.

[16] Notice that *ceq* being reflexive does not follow from *ceq* being symmetric and transitive.

[17] *Given an action symbol A and distinctive state ds, $A(ds) = ds'$ if the schema $\langle ds, A, ds' \rangle$ has been observed, otherwise, $A(ds) = \perp$. Moreover, $A(\perp) = \perp$. The definition is then extended to action sequences in the standard way. Notice that $A(ds)$ being well-defined relies on our assumption that actions are deterministic (axiom 4.22).*

Definition 2

A set of experiences E is **complete** whenever

$$E \models \forall a, ds \exists ds' \langle ds, a, ds' \rangle \quad .$$

$\{$*end of definition*$\}$

Theorem 2 *Let E be a complete set of experiences. Let* CT(E) *be defined as follows:*

$$\begin{array}{l} CT(E) \ = \\ \quad COMPLETION(E) \ , \\ \quad Axioms \ 4.1 - 4.11 \ , \\ \quad \langle ds, a, ds' \rangle \wedge \langle ds, a, ds'' \rangle \rightarrow ds' = ds'', \qquad (Axiom \ 4.22) \\ \quad CEQ\_block \ = \\ \quad \{ \ max \ \ ceq : \\ \quad\ \ ceq(ds_1, ds_2) \rightarrow View(ds_1, v) \equiv View(ds_2, v), \\ \quad\ \ ceq(ds_1, ds_2) \wedge \langle \ ds_1, a, ds_1' \ \rangle \wedge \langle \ ds_2, a, ds_2' \ \rangle \rightarrow ceq(ds_1', ds_2') \\ \quad \} \end{array}$$

Then the predicate ceq is an equivalence relation.

Proof. See appendix E (page 232).

When a set of experiences is complete the predicate *ceq* captures the idea that two distinctive states are the same if they render the same views under any sequence of actions. Assume that E is complete and let $A = a_1, \ldots, a_n$ denote a sequence of actions. Since E is complete, $A(s) \neq \perp$ and the formula $View(A(ds), v)$ makes sense.

Theorem 3 *Let E be a complete set of experiences. Then,*

$$ceq(ds_1, ds_2) \equiv \forall A, v \ \ [View(A(ds_1), v) \equiv View(A(ds_2), v)] \ \ .$$

Proof. *See appendix E (page 234).*

The problem of distinguishing environment states by outputs (views) and inputs (actions) has been studied in the framework of automata theory [Angluin, 1978, Gold, 1978, Rivest and Schapire, 1987, Basye *et al.*, 1995]. In this framework, the problem we address is the one of finding the minimum automaton (w.r.t. the number of states) consistent with a given set of input/output pairs. Without any particular assumptions about the environment or the agent's perceptual abilities, the problem of finding this smallest automaton is NP-complete ([Angluin, 1978, Gold, 1978]). The reader is referred to [Basye *et al.*, 1995] for an example of how to use automata to model dynamical systems.

Example 4

Consider the set of experiences *E* as in example 2 (page 23) (see figure 4.3a):

$$\begin{array}{l} Action\_type(ml, travel)\ , \\ CS(s1, a, ml, b)\ ,\ CS(s2, b, ml, c)\ , \\ View(a, v+)\ ,\ View(b, v+)\ ,\ View(c, v+)\ . \end{array}$$

Since the same view is experienced at *a*, *b* and *c*, the extent of ceq is maximized by declaring $ceq = true$ (i.e. $\forall x, y\ ceq(x, y)$). Notice that axiom (4.25) is trivially satisfied since no action has been executed at *c*.

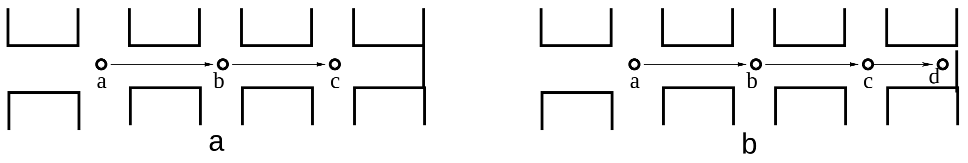


Figure 4.3: (a) Distinctive states *a*, *b* and *c* cannot be distinguished at the causal level. Topological information is needed in order to distinguish them. (see text) (b) *a*, *b* and *c* are distinguished given the new information $\langle c, travel, d\rangle$.

Though *a*, *b* and *c* were experienced at different states environment states, at the causal level they are declared as indistinguishable. This happens because neither the actions nor the views provide enough information to distinguish them. By using topological information (i.e. the concepts of $path$ and $place$ in chapter 5) we will be able to distinguish these distinctive states (see example 8, page 47).

Suppose the agent continues exploring the environment and gets the new information

$$View(d, v\ \sqsupset),\ CS(s3, c, ml, d),$$

as suggested in figure 4.3b. In virtue of lemma 1, it can be seen that $ceq(ds, ds') \equiv ds = ds'$, and consequently all distinctive states refer to different environment states.

$\{$*end of example*$\}$

As shown in the following example, **there could exist different non-isomorphic models that maximize the extent of the predicate** *ceq*.

Example 5

Let E be the set defined by the following formulae:

$$Action\_type(ml, travel) \ ,$$
$$CS(s1, a, ml, b) \ , \ CS(s2, c, ml, d) \ .$$
$$View(a, v) \ , \ View(b, v) \ , \ View(c, v) \ , \ View(d, v1)$$

There are two different models for *ceq*. In one of them, $ceq(a, b) \wedge \neg ceq(b, c)$ is the case, and in the other one $\neg ceq(a, b) \wedge ceq(b, c)$ is the case. Notice that in virtue of axiom 4.25, in both of these models $\neg ceq(a, c)$ is the case.
$\{$*end of example*$\}$

Different models of $CT(E)$ generally arise when the set of experiences E is incomplete (i.e. the agent has not completely explore the environment) or weak sensors inputs at different environment states are classified as the same view.

Example 6

Consider the environment depicted in figure 4.4. The agent visits the different distinctive states as suggested by their numbers in the figure. The same travel action ml is performed when traveling from a corner to the intersection (i.e $\langle 1, ml, 2 \rangle$) and viceversa (i.e. $\langle 4, ml, 5 \rangle$). A $turn\_around$ action is performed when reaching a corner (i.e. $\langle 3, turn\_around, 4 \rangle$,$\langle 7, turn\_around, 8 \rangle$, etc.). Assume that the different corners have the same views (i.e. view(1) = view(4) = view(8), view(3)= view(7) = view(11)), and views associated with the other distinctive states are different.

Three models of $CT(E)$ can be associated with the exploration E of the T-environment:

1. Model 1: $ceq(8, 12), ceq(12, 8), ceq(x, x)$.[18]

2. Model 2: $ceq(1, 12), ceq(12, 1), ceq(x, x)$.

3. Model 3: $ceq(4, 12), ceq(12, 4), ceq(3, 11), ceq(11, 3), ceq(2, 10), ceq(10, 2), ceq(x, x)$.

[18]The extent of *ceq* in model 1 is defined by $\{(8, 12), (12, 8)\} \cup \{(x, x) \ : x = 1, \ldots, 12\}$.

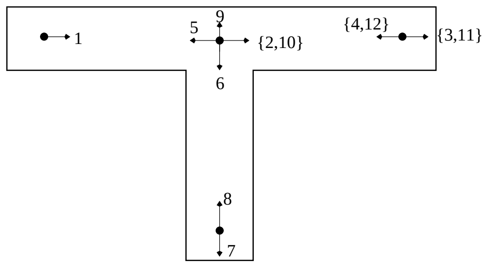


Figure 4.4: The agent visits the different distinctive states in the order suggested by their numbers. The same view occurs at the different corners (i.e view(1)= view(4) = view (8)). Three different causal models can be associated with the agent exploration of this T-environment (see text).

Notice that in all the models above, $\neg ceq(1,4)$, $\neg ceq(1,8)$, $\neg ceq(4,8)$. For instance, from $\langle\, 1, ml, 2\,\rangle$,$\langle\, 4, ml, 5\,\rangle$, and $view(2) \neq view(5)$ we conclude that $\neg ceq(1,4)$. Notice that although dstate 12 is at the same environment state as dstate 4, it is possible that $ceq(1,12)$ or $ceq(8,12)$. This is the case since no action has been performed at dstate 12.

Notice that the models of $CT(E)$ are maximal with respect to the set inclusion for ceq. The number of elements in the possible extents of ceq could vary, and consequently the number of different environment states represented by the models of $CT(E)$. For instance, the three models above represent 11, 11 and 10 environment states respectively.[19]

Finally, notice that all the models above are possible since at the causal level turn and travel actions do not bear any spatial meaning. Should we consider topological information, only model 3 above will be possible (see example 15, page 53). {*end of example*}

4.3 The SSH causal graph

The SSH causal graph associated with a set of experiences E is the labeled graph $\langle Nodes, Edges, Labels\rangle$ such that:

- Nodes = $DS(E)/ceq$.
- Labels = $A(E)$.
- Edges = $\{([ds], a, [ds]') \ : \ COMPLETION(E) \models \langle ds, a, ds'\rangle \ \}$.

[19]See example 34 (page 148) for a program trace illustrating how the causal models change as the exploration of the T-environment proceeds.

where $DS(E)/ceq$ denotes the set of equivalence classes of *DS(E)* modulo *ceq*, and $[ds]$ denotes the equivalence class of ds given *ceq*.

Example 7

The SSH causal graph associated with the set of experiences in example 4 is shown in figure 4.5. Notice that the causal graph and the view graph can be isomorphic. However, in general, the causal graph makes states distinctions that cannot be derived from views alone.

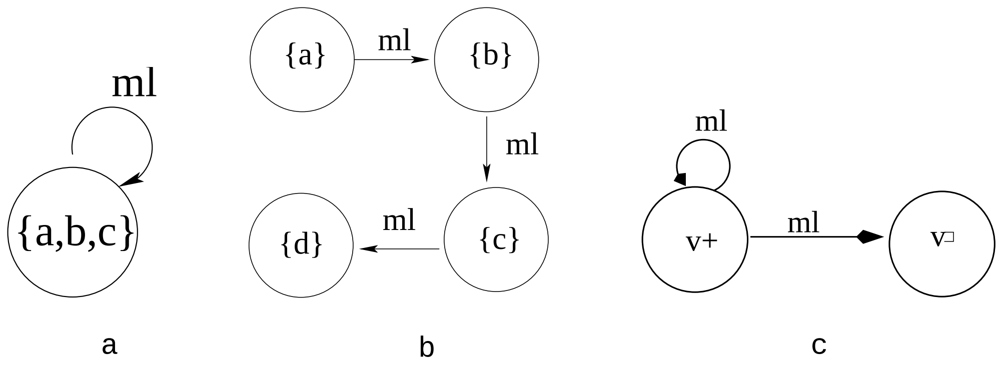


Figure 4.5: SSH causal graph associated with the set of experiences in figures 4a and 4b. (c) view graph associated with the set of experiences in figure 4b. The causal graph in *a* is isomorphic to the view graph associated with the experiences in figures 4a (Example 3, page 25).

{*end of example*}

4.4 Calculating the models of CT(E)

The Herbrand models of $CT(E)$ are in a one to one correspondence with the answer sets [Gelfond and Lifschitz, 1991] of the logic program in figure 4.6.[20] In this program, the X and Y variables range over distinctive states and the variable V ranges over views in E. The sets of rules 4.26 and 4.27 are the facts corresponding to the agent's experiences. Rule 4.28 states that an answer set of the program should be *complete* with respect to ceq. Rules 4.29-4.31 require ceq to be an equivalence class. Rules 4.31 and 4.32 are the counterpart of axiom 4.24. Rule 4.34 is the counterpart of axiom 4.25. In order to define the maximality condition of ceq, the auxiliary predicate $p(X, Y, X1, Y1)$ is introduced. This predicate reads as *"If X and Y were the same, then $X1$ and $Y1$ would be the same"*. The predicate

[20]See appendix D, page 226, for the definition and properties of answer sets. See appendix F, page 236, for a proof of the correctness of the logic program in Figure 4.6.

$$\{cs(ds, a, ds') \leftarrow . \quad : cs(ds, a, ds') \in E\} \tag{4.26}$$
$$\{view(ds, v) \leftarrow . \quad : view(ds, v) \in E\} \tag{4.27}$$
$$ceq(X, Y), \neg ceq(X, Y) \leftarrow \quad .$$
$$p(X, Y, X, Y) \leftarrow .$$
$$p(X, Y, X2, Y1) \leftarrow p(X, Y, X1, Y1), ceq(X1, X2).$$
$$p(X, Y, X1, Y2) \leftarrow p(X, Y, X1, Y1), ceq(Y1, Y2).$$
$$p(X, Y, X2, Y2) \leftarrow p(X, Y, X1, Y1), cs(X1, A, X2), cs(Y1, A, Y2).$$
$$p(X, Y, Y1, X1) \leftarrow p(X, Y, X1, Y1).$$
$$p(X, Y, X1, Y2) \leftarrow p(X, Y, X1, Y1), p(X, Y, Y1, Y2).$$

$$dist(X, Y) \leftarrow p(X, Y, X1, Y1), view(X1, V), not\ view(Y1, V).$$
$$dist(X, Y) \leftarrow p(X, Y, X1, Y1), not\ view(X1, V), view(Y1, V).$$

$$ceq(X, Y); \neg ceq(X, Y) \leftarrow . \tag{4.28}$$
$$\leftarrow not\ ceq(X, X). \tag{4.29}$$
$$\leftarrow ceq(X, Y), not\ ceq(Y, X). \tag{4.30}$$
$$\leftarrow ceq(X, Y), ceq(Y, Z), not\ ceq(X, Z). \tag{4.31}$$
$$\leftarrow ceq(X, Y), view(X, V), not\ view(Y, V). \tag{4.32}$$
$$\leftarrow ceq(X, Y), not\ view(X, V), view(Y, V). \tag{4.33}$$
$$\leftarrow not\ ceq(X1, Y1), ceq(X, Y), cs(X, A, X1), cs(Y, A, Y1). \tag{4.34}$$

$$\leftarrow not\ ceq(X, Y), not\ dist(X, Y). \tag{4.35}$$

Figure 4.6: Logic program associated with CT(E).

$dist(X, Y)$ defines when distinctive states X and Y are distinguishable. Constraint 4.35 establishes the maximality condition on ceq: $ceq(X, Y)$ should be the case unless X and Y are distinguishable.[21] In section 9.4.1 (page 147) we illustrate the use of the program for a simulated robot navigating a T-like environment.

[21]We have implemented this logic program in Smodels [Niemelä and Simons, 1997]. In the implementation, one has to add variable domain restrictions to the different rules. For example, rule

$$ceq(X, Y), \neg ceq(X, Y) \leftarrow \quad .$$

becomes

$$ceq(X, Y), \neg ceq(X, Y) \leftarrow \; dstate(X), dstate(Y)$$

where $dstate$ is our predicate to identify the sort of distinctive states.

4.5 Summary

SSH schemas summarize the continuous interactions of the agent in the environment. This is done by storing the initial and final distinctive states (and their corresponding views) for any action execution. Schemas can then later be used for directing the agent behavior, planning, or distinguishing distinctive states sharing a same view.

By considering only the views associated with the initial and final distinctive states of a schema, we defined the *SSH view graph* (section 4.2.2, page 24), which relates different views by actions linking them. When the agent can discern most distinctive states by their views, the view graph can be used for planning routes between different distinctive states. The view graph representation is consistent with human navigation abilities [Mallot and Gillner, 2000, Steck and Mallot, 2000].

By considering actions as well as views, the agent can further distinguish distinctive states. Two distinctive states, ds and ds', are distinguishable, if there is a sequence of actions that when executed at ds renders a different sequence of views than when it is executed at ds'. "Spatial properties" are not associated with actions when used to distinguish distinctive states. This may prevent the agent from differentiating distinctive states that correspond to different environment states. In the next chapter we show how action's "spatial properties" can be used to create a different ontology from the causal one - that of *places* and *paths*- which in turn can be used to further differentiate distinctive states.

In section 4.2.3 (page 26) we defined the predicate ceq which is the case for distinctive states that are not distinguishable by actions and views. We investigated whether the axioms defining ceq (the CEQ_block) were necessary and stated conditions under which they can be simplified (see appendix E, Page 229). We then defined the *SSH causal graph* whose nodes are classes of distinctive states (classes w.r.t ceq). This representation is akin to the view graph although it imposes further refinement in the set of environment states that are consistent with the agent experiences. The problem of identifying the minimum set of distinctive states consistent with the agent's experiences is equivalent to the one of identifying the minimum automata consistent with a set of input/output pairs. This problem turns out to be NP-complete when no special properties about the actions, views, or the environment are assumed [Angluin, 1978, Gold, 1978, Basye *et al.*, 1995].

Finally, in section 4.4 we defined a logic program whose answer sets are in an one to one correspondence with the models of the SSH causal theory. In appendix F we present a proof of this claim.

Chapter 5

Topological Level

Relations among the distinctive states and trajectories defined by the control level, and among their summaries as schemas at the causal level, are effectively described by the topological network. At the SSH topological level the ontology consists of *places*, *paths* and *regions*, with connectivity and containment relations among them.

Roughly speaking, a *place* corresponds to a set of distinctive states linked by turn with no travel actions. Similarly, a *path* corresponds to a set of distinctive states linked by travel with no turn actions (except for *turn around* actions). Places and paths are created by *abduction*, positing the minimal set of places, paths, and regions required to explain the available observations (i.e. a set of schemas).

Grouping places into *regions* allows an agent to reason efficiently about its spatial knowledge. Regions themselves can be grouped to form new regions. In chapter 8, we will study how to formally include this hierarchy of regions in the SSH. In the next chapter we define *boundary regions* associated with paths. Informally, a path has associated three disjoint regions: the set of places in the path, the set of places to the left of the path, and the set of places to the right of the path. Boundary regions allow the agent to distinguish distinctive states, for two distinctive states can be considered different if they are in different boundary regions (see example 23, page 84).

The construction of the topological map is usually described as an abduction process [Kuipers and Byun, 1988, Kuipers and Byun, 1991, Kuipers *et al.*, 1993, Kuipers, 1996, Kuipers, 2000]. This abduction process is defined in procedural terms inspired by the current implementation of the SSH. In this chapter we define the circumscriptive theory associated with the SSH topological level. The models associated with this theory define the *preferred models* to be constructed by the SSH abduction process.

We will present two versions of the SSH topological theory. In the first version (section 5.1) we make the assumption that paths cannot be self intersecting nor two different paths can converge to a same distinctive state (see figure 5.1). In the second version (section 5.5), a general topological theory makes these assumptions defeasible and embodies the default that "normally, self-intersecting or convergent paths do not exist". While the later theory is more general, it is harder to calculate the consequences of its different defaults. However, both theories are related in that any map with respect to the simpler theory is a map with respect to the general theory.

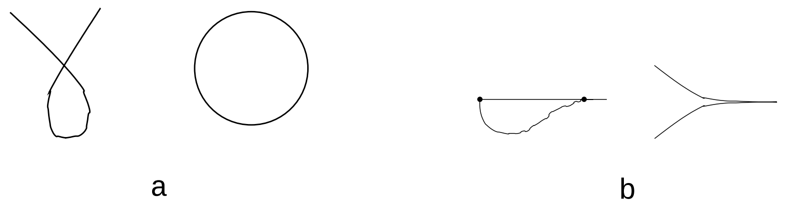


Figure 5.1: (a) Self intersecting paths. (b) Convergent paths.

This chapter is organized as follows. Section 5.1 formally defines the models associated with the SSH topological map. Section 5.3 illustrates the case when the topological map is not unique. In section 5.4 we show how to simplify the theory under the assumption that views uniquely identify distinctive states. We present the general SSH topological theory in section 5.5. An algorithm to calculate the models of the topological theory is defined in section 5.6. Proofs for the different claims made in this chapter are presented in appendix B (page 214).

5.1 The SSH Topological Theory

As in chapter 4, we assume that the agent's experiences in the environment are given by a formula E (see section 4.2, page 21). We are to define the SSH topological theory, $TT(E)$, associated with E. The language of this theory is a sorted language with sorts for *places*, *paths* and *path directions*. The main purpose of the topological theory $TT(E)$ is to minimize the set of topological paths and topological places consistent with the given experiences E.

Within the sort of places, we distinguish between *topological places* and *regions*.

A topological place is a set of distinctive states linked by turn actions. A region is a set of places. We use the predicates *tpath* and *is_region* to identify these subsorts. A path defines an order relation among places connected by travel with no turn actions. They play the role of streets in a city layout. Among paths, *topological paths* correspond to those paths whose places are topological places. We use the predicate *tpath* to identify these paths. Other paths could exists whose places are regions. We use the predicate *route* to identify these paths.

The distinctions between topological places and regions and between topological paths and routes will allow the agent to learn from its own experiences as well as from given information. For example, the agent could learn about places and paths by being told, or by "reading" a map. Our focus in this chapter is on describing the map learned from actual experiences. We postpone the study of regions until chapters 6 and 8.

We require the sorts of $places$ and $paths$ to be infinite. This is not to say that the SSH topological map has infinite number of $places$ or $paths$. As explained in section 5.3 (page 59), given a model of our theory, the SSH topological map corresponds to the submodel obtained by restricting the different predicates to *topological places*, *regions*, *topological paths* and *routes*. Since *topological places* are identified with set of distinctive states and *topological paths* are identified with sequences of distinctive states, the topological map associated with a finite set of schemas (and so a finite set of distinctive states) has a finite number of *topological places* and *topological paths*.[1] Our requirement of infinite $places$ and $paths$ allow us to compare any two models of the theory. Example 22 (page 67) illustrates the use of this requirement.

A path has associated two directions, *pos* and *neg*. A path direction provides a frame of reference to establish the order in which places in a path are arranged or whether a place is to the right or left of the path (see boundary regions in chapter 6). The sort of path directions is completely defined by *pos* and *neg*:[2]

$$pos \neq neg \ , \tag{5.1}$$

$$\forall dir \quad \{dir = pos \vee dir = neg\} \quad . \tag{5.2}$$

The language of the SSH topological level includes the following predicates:[3]

1. *teq(ds,ds')*: *distinctive states* ds and ds' are *topologically* indistinguishable.

[1]See proof in appendix B, theorem B, page 216.

[2]For a direction dir, $-dir$ is defined such that $-pos = neg$ and $-neg = pos$.

[3]Later we will make precise the informal meaning of these predicates.

2. *tplace(p)* : *place* p is a *topological* place.

3. *is_region(r)*: *place* r is a *region*.

4. *tpath(pa)* : *path* pa is a *topological* path.

5. *route(pa)*: *path* pa is a *route*.

6. *on(pa,p)*: *place* p is on *path* pa.

7. *order(pa,dir,p,q)* : *place* p is before *place* q, when facing direction *dir* on path *pa*.

8. *at(ds,p)* : *distinctive state* ds is at *place* p.

9. *along(ds,pa,dir)*: *distinctive state* ds is along *path* pa in direction *dir*.

The SSH topological theory associated with E, **TT(E)**, is the following nested abnormality theory (NATs) [Lifschitz, 1995] (see appendix C, page 219):[4][5]

$$
\begin{aligned}
&TT(E) \ = && (5.3)\\
&\quad there\ exist\ infinitely\ many\ places\ ,\\
&\quad there\ exist\ infinitely\ many\ paths\ ,\\
&\quad \neg\exists p\,[tplace(p) \wedge is\_region(p)]\,, && (5.4)\\
&\quad \neg\exists pa\,[tpath(pa) \wedge route(pa)]\ , && (5.5)\\
&\quad Axioms\ \ 5.1 - 5.2\ ,\\
&\quad COMPLETION(E)\ ,\\
&\quad Axioms\ 4.1 - 4.11\ ,\\
&\quad \langle ds, a, ds'\rangle \wedge \langle ds, a, ds''\rangle \rightarrow ds' = ds'', \qquad (Axiom\ 4.22)\\
&\quad T\_block\ ,\\
&\quad \{\ min\ is\_region,\ min\ route\ :\\
&\quad\ \ AT\_block\\
&\quad \}
\end{aligned}
$$

Axiom 5.4 says that topological places and regions are two subsorts of places. Similarly, axiom 5.5 says that topological paths and routes are two subsorts of paths. The minimization criteria establishes that we are interested in models where the universe of these subsorts are as small as possible in order to explain E. Further requirements on paths

[4]A block of the form $\{C_1, \ldots, C_n,\ min\, P_1, \ldots, min\, P_k\ :\ A_1, \ldots, A_m\}$ denotes the set of blocks $\{C_1, \ldots, C_n,\ min\, P_1\ :\ A_1, \ldots, A_m\}, \ldots, \{C_1, \ldots, C_n,\ min\, P_k\ :\ A_1, \ldots, A_m\}$.

[5]The condition that the sorts of places and paths are infinite can be formalized requiring the existence of a bijection between these sorts and the natural numbers.

and places will be given by the block *AT_block*. In particular, since in this chapter we do not define axioms using regions or routes,[6] it is the case that the following two default conclusions follow from $TT(E)$:

$$\forall p\ \neg is\_region(p) \quad , \quad \forall pa\ \neg route(pa)\ .$$

That is, in this chapter all places and paths we deal with are topological places and topological paths respectively.

The block **T_block** defines the properties of the predicates $\widehat{turn}$, $\widehat{travel}$, and $\vec{travel}$. $\widehat{turn}$ is the equivalence closure of the schemas $\langle\cdot, turn, \cdot\rangle$; $\widehat{travel}$ and $\vec{travel}$ are the equivalence and transitive closure of the schemas $\langle\cdot, travel, \cdot\rangle$ respectively.

$$\begin{aligned}
&T\_block\ = \\
&\quad \{\ min\ \ \widehat{turn}, min\ \ \widehat{travel}, min\ \ \vec{travel} : \\
&\quad\ \ \langle ds, turn, ds'\rangle \rightarrow \widehat{turn}(ds, ds'), \\
&\quad\ \ \langle ds, travel, ds'\rangle \rightarrow \widehat{travel}(ds, ds') \wedge \vec{travel}(ds, ds'), \\
&\\
&\quad\ \ \widehat{turn}(ds, ds), \\
&\quad\ \ \widehat{turn}(ds, ds') \rightarrow \widehat{turn}(ds', ds), \\
&\quad\ \ \widehat{turn}(ds, ds') \wedge \widehat{turn}(ds', ds'') \rightarrow \widehat{turn}(ds, ds''), \\
&\\
&\quad\ \ \widehat{travel}(ds, ds), \\
&\quad\ \ \widehat{travel}(ds, ds') \rightarrow \widehat{travel}(ds', ds), \\
&\quad\ \ \widehat{travel}(ds, ds') \wedge \widehat{travel}(ds', dr) \rightarrow \widehat{travel}(ds, dr), \\
&\\
&\quad\ \ \vec{travel}(ds, ds') \wedge \vec{travel}(ds', ds'') \rightarrow \vec{travel}(ds, ds'') \\
&\quad \}
\end{aligned} \tag{5.6}$$

The block **AT_block** is the heart of our theory. It defines how the agent groups distinctive states into *places*, and how *places* are ordered by *paths*. The purpose of this block is to define the extent of the predicates *tpath*, *tplace*, *at*, *along*, *order*, *on* and *teq*. The block has associated the circumscription policy[7]

$$\textbf{circ}\ \ tpath \succ tplace\ \ \textbf{var}\ \ \vec{SSHpred}$$

[6]The predicates *is_region* and *route* do not occur in *AT_block*. In chapters 6 and 8 we will add axioms to *AT_block* including these predicates.

[7]The symbol $\succ$ indicates prioritized circumscription (see [Lifschitz, 1994] section 7.2, also appendix C, page 219).

where $\vec{SSHpred}$ stands for the tuple of predicates $\langle \mathbf{at}, \mathbf{along}, \mathbf{order}, \mathbf{on}, \mathbf{teq}, turn\_eq, travel\_eq, \rangle$.[8] This circumscription policy states that a minimum set of topological paths is preferred to a minimum set of topological places. The block AT_block is defined as follows:[9] [10]

$$
\begin{aligned}
& AT\_block \ = && (5.7)\\
& \quad \{ \ max \ teq \ : \\
& \quad\quad teq \ is \ an \ equivalence \ relation \ , && (5.8)\\
& \quad\quad teq(ds_1, ds_2) \rightarrow View(ds_1, v) \equiv View(ds_2, v), && (5.9)\\
& \quad\quad teq(ds_1, ds_2) \wedge \langle \, ds_1, a, ds'_1 \, \rangle \wedge \langle \, ds_2, a, ds'_2 \, \rangle \rightarrow teq(ds'_1, ds'_2) && (5.10)\\
& \quad\quad teq(ds_1, ds_2) \rightarrow \forall p \, [at(ds_1, p) \equiv at(ds_2, p)] \wedge && (5.11)\\
& \quad\quad\quad\quad\quad\quad \forall pa, dir \, [along(ds_1, pa, dir) \equiv along(ds_2, pa, dir)] \\
& \quad\quad \langle ds, Turn, ds' \rangle \rightarrow \neg teq(ds, ds') \ , && (5.12)\\
& \quad\quad \langle ds, turnAround, ds' \rangle \wedge \langle ds, turnAround, ds'' \rangle \rightarrow teq(ds', ds'') \ , && (5.13)\\
& \quad\quad \langle \, ds_1, turnAround, ds_2 \, \rangle \wedge \langle \, ds_2, turnAround, ds_3 \, \rangle \rightarrow teq(ds_1, ds_3) \ , && (5.14)\\
& \\
& \quad\quad at(ds, p) \rightarrow tplace(p), && (5.15)\\
& \quad\quad \exists! p \, at(ds, p), && (5.16)\\
& \quad\quad turn\_eq(ds_1, ds_2) \equiv \forall p \, [at(ds_1, p) \equiv at(ds_2, p)] \, , && (5.17)\\
& \quad\quad \{ min \ \ turn\_eq : && (5.18)\\
& \quad\quad\quad teq(ds_1, ds_2) \wedge teq(ds_3, ds_4) \wedge \widehat{turn}(ds_2, ds_3) \rightarrow turn\_eq(ds_1, ds_4), \\
& \quad\quad\quad turn\_eq(ds_1, ds_2) \wedge turn\_eq(ds_2, ds_3) \rightarrow turn\_eq(ds_1, ds_3) \\
& \quad\quad \} \\
& \quad\quad along(ds, pa, dir) \rightarrow tpath(pa), && (5.19)\\
& \quad\quad \{ \ min \ along : && (5.20)
\end{aligned}
$$

[8]The predicates $travel\_eq$ and $turn\_eq$ are "auxiliary" predicates used in our topological theory. Although they are completely defined in terms of teq, $\widehat{turn}$ and $\widehat{travel}$, they need to vary in the circumscription policy. In order to see why this is the case consider the statements $CIRC\,[(Q \rightarrow F) \wedge P \equiv Q; F; Q]$ and $CIRC\,[(Q \rightarrow F) \wedge P \equiv Q; F; Q, P]$. In both cases F is circumscribed given the theory $(Q \rightarrow F) \wedge P \equiv Q$. Notice that P is completely defined in terms of Q. However, it is the case that

$$
\begin{aligned}
& CIRC\,[(Q \rightarrow F) \wedge P \equiv Q; F; Q] \equiv [P \equiv Q \equiv F] \ , \\
& CIRC\,[(Q \rightarrow F) \wedge P \equiv Q; F; Q, P] \equiv [P \equiv Q \equiv F \equiv false] \ .
\end{aligned}
$$

Although P is defined in terms of Q, it is necessary to vary both in order to make a stronger assertion about the minimality of F.

[9]Figure 5.2, page 43, summarizes the dependencies among the predicates defined by the AT_block.

[10]See page 224 for NAT's notation.

$$\langle\, ds, travel, ds'\,\rangle \rightarrow \exists pa,\ dir\ [along(ds,pa,dir) \wedge along(ds',pa,dir)]\,, \tag{5.21}$$

$$\langle ds, turnAround, ds'\rangle \rightarrow along(ds,pa,dir) \equiv along(ds',pa,-dir), \tag{5.22}$$

$$teq(ds_1,ds_2) \rightarrow along(ds_1,pa,dir) \equiv along(ds_2,pa,dir) \tag{5.23}$$

$\}$

$$along(ds,pa,dir) \wedge along(ds,pa1,dir1) \rightarrow pa = pa1 \wedge dir = dir1, \tag{5.24}$$

$$at(ds_1,p) \wedge at(ds_2,p) \wedge along(ds_1,pa,dir) \wedge along(ds_2,pa,dir) \rightarrow teq(ds_1,ds_2), \tag{5.25}$$

$$[\langle ds, turn\_desc, ds'\rangle\ \wedge\ turn\_desc \neq turnAround\ \wedge \tag{5.26}$$
$$along(ds,pa,dir) \wedge along(ds',pa1,dir1)] \rightarrow pa \neq pa1,$$

$$\{\ min\ order: \tag{5.27}$$

$$[\langle\, ds, travel, ds'\,\rangle \wedge at(ds,p) \wedge at(ds',q) \wedge \tag{5.28}$$
$$along(ds,pa,dir) \wedge along(ds',pa,dir)] \rightarrow order(pa,dir,p,q),$$

$$order(pa,pos,p,q) \equiv order(pa,neg,q,p), \tag{5.29}$$

$$order(pa,dir,p,q) \wedge order(pa,dir,q,r) \rightarrow order(pa,dir,p,r) \tag{5.30}$$

$\}$

$$\neg order(pa,dir,p,p), \tag{5.31}$$

$$\{min\ on:\ at(ds,p) \wedge along(ds,pa,dir) \rightarrow on(pa,p)\ \} \tag{5.32}$$

$$on(pa,p) \wedge on(pa,q) \wedge tpath(pa) \rightarrow \tag{5.33}$$
$$\exists ds_1, dir_1, ds_2, dir_2\ [at(ds_1,p) \wedge along(ds_1,pa,dir_1) \wedge at(ds_2,q) \wedge$$
$$along(ds_2,pa,dir_2) \wedge travel\_eq(ds_1,ds_2)]\,,$$

$$\{min\ \ travel\_eq: \tag{5.34}$$

$$\widehat{travel}(ds_1,ds_2) \rightarrow travel\_eq(ds_1,ds_2), \tag{5.35}$$

$$\langle\, ds_1, turnAround, ds_2\,\rangle \rightarrow travel\_eq(ds_1,ds_2) \wedge travel\_eq(ds_2,ds_1) \tag{5.36}$$

$$teq(ds_1,ds_2) \wedge teq(ds_3,ds_4) \wedge travel\_eq(ds_2,ds_3) \rightarrow travel\_eq(ds_1,ds_4),$$

$$travel\_eq(ds_1,ds_2) \wedge travel\_eq(ds_2,ds_3) \rightarrow travel\_eq(ds_1,ds_3)$$

$\}$

$$\mathbf{circ}\ \ tpath \succ tplace\ \ \mathbf{var}\ \ \vec{SSHpred} \tag{5.37}$$

$\}$

We discuss these axioms in turn.

Predicate *teq* stands for *topologically equal*. Whenever $teq(ds_1,ds_2)$ is the case, we can consider ds_1 and ds_2 as denoting the same environment state. If $teq(ds_1,ds_2)$ is

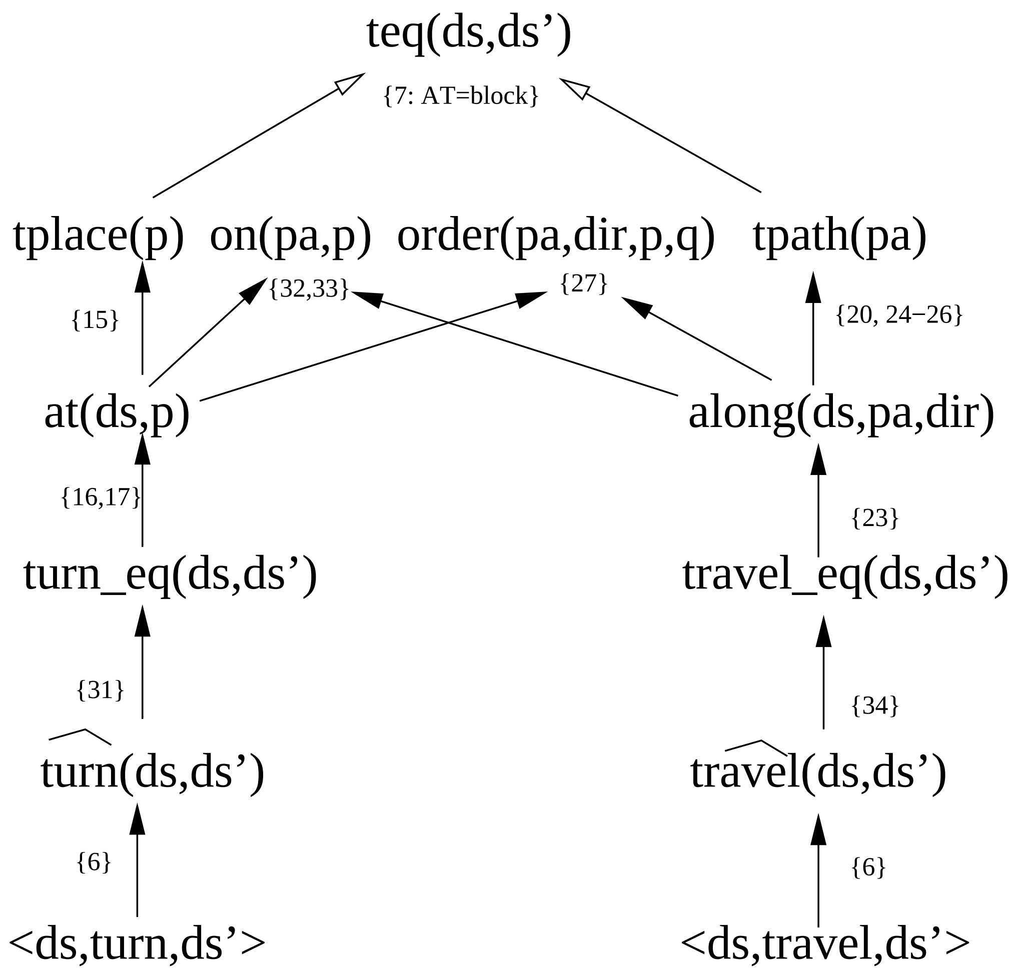


Figure 5.2: Dependency among predicates in $TT(E)$. Labels on the graph's arrows refer to the axioms relating the predicates pointed by the arrows.
Distinctive states related by turns modulo teq ($turn\_eq$) must be at the same topological place ($tplace$). Distinctive states related by travels modulo teq ($travel\_eq$) are $along$ the same topological path ($tpaths$). Knowing at which places and along which paths distinctive state are, determines what places are on what paths. The order of places on a path is derived from travels among distinctive states along a path.
Since the extents of $travel\_eq$ and $turn\_eq$ must be defined in order to determine places and paths, one needs to know what distinctive states are teq. The arrows pointing to teq on the top of the diagram indicate that among the possible interpretations for teq, the preferred models of the theory select those that lead to a map where a minimum set of paths and places are needed to explain the schemas at the bottom of the diagram. An efficient search algorithm can be defined (section 5.6, page 71) to explore the space of teq possibilities given a set of schemas.

the case, ds_1 and ds_2 cannot be distinguished by views and actions (axioms 5.9 and 5.10), they are at the same place and along the same path direction.[11]

Axiom 5.12 states that a $turn$ action takes the agent from one distinctive state to a different one. In particular we assume that a schema of the form $\langle ds, Turn, ds \rangle$ is not included in the agent's experiences. Axiom 5.13 states that there is a unique (modulo teq) distinctive state resulting from performing a turn around action. After two turn around actions the agent is back to the same dstate (Axiom 5.14). Turn around actions are special since they link distinctive states along the same path but in opposite directions (axiom 5.22).

Axioms 5.16-5.17 state how the agent groups distinctive states into places. Every distinctive state is at a unique topological place (axiom 5.16). Whenever the agent $turns$, it stays at the same topological place (axiom 5.17). Distinctive states grouped into a topological place should be $turn$ connected (modulo teq) (axiom 5.17).[12]

Travel actions among distinctive states are abstracted to topological paths connecting the places associated with such distinctive states (axioms 5.21 and 5.28). Travel axioms are explained in terms of the two related predicates, $along$ and $order$. Both of these predicates are the minimum ones explaining travel actions and satisfying other properties included in their respective blocks 5.20 and 5.27.

Block 5.20 defines the predicate $along$. Whenever an agent *turns around*, it stays in the same path but facing the opposite path's direction (axiom 5.22). Axiom 5.23 is a trivial consequence of the definition of teq but it needs to be included in the block so that the interpretation of $along$ has tuples other than the ones explicitly derived from schemas (see example 10, page 49).

There are further restrictions on the properties of $along$. For instance, a distinctive state is along at most one path (axiom 5.24). Since axiom 5.24 provides "negative" information about $along$, it does not need to be included in block 5.20 (see proposition 4 in [Lifschitz, 1994]). Axiom 5.24 prevents the existence of different paths that converge to the same distinctive state (see example 18, page 57). In section 5.5 we will make this axiom a default.

[11]It is possible that the maximization of teq in the AT_block implies that teq is an equivalence relation without explicitly requiring so. However at this point we do not have a proof for this claim.

[12]Block 5.18 states that the predicate $turn\_eq$ corresponds to the relation $\widehat{turn}$ modulo teq.

Turn actions other than *turnAround* change the path the initial and final distinctive states linked by the action are along (axiom 5.26). This axiom allows the agent to conclude the existence of different paths once it turns right or left at a place (see Example 13, Page 51). This axiom prevents the existence of self-intersecting paths as illustrated in figure 5.1. In section 5.5 we will make this axiom a default.

Block 5.27 defines the predicate $order$. In addition to explaining travel actions, $order$ defines an order among the places on a path satisfying the following two properties: i) the order of places in a given path direction is the inverse of the order of places in the other path direction (axiom 5.29), and ii), the order of places in a path is transitive (axiom 5.30).

There are further restrictions on the properties of $order$: i) the order of places in a path should be non-reflexive (axiom 5.31), and ii) the agent has to have traveled among the places on the same path (axiom 5.33). Since these requirements provide "negative" information about $order$, they do not need to be included in block 5.27 (see proposition 4 in [Lifschitz, 1994]). Notice that we rule out the existence of circular paths (axiom 5.31). In section 5.5 we will make this axiom a default.

Axiom 5.33 requires the agent to have traveled among the places on a same path. $travel\_eq$ defines when two distinctive states are linked by travel actions without turns (except for $TurnAround$ actions) (see block 5.34). Example 14 illustrates how by using $travel\_eq$ the agent can minimize the set of topological paths.

The SSH topological theory $TT(E)$ describes those places, $tplaces$, and paths, $tpaths$, that capture the intended meaning of $turn$ and $travel$ actions. This is done in such a way that a minimum number of topological places and topological paths are identified. The block AT_block formally states this minimality criterion. A minimum set of topological paths is identified even at the cost of increasing the set of topological places. Figure 5.2 shows the dependency among the different predicates in $TT(E)$.

Remark. Axiom 5.24 implies a stronger version of axioms 5.21 and 5.28, namely:

$$\begin{aligned} &at(ds,p) \wedge at(ds',q) \wedge \vec{travel}(ds,ds') \rightarrow \\ &\quad \exists! pa,\ dir\ [order(pa,dir,p,q) \wedge along(ds,pa,dir) \wedge along(ds',pa,dir)]\,. \end{aligned}$$

Using the above statement in combination with axiom 5.31 we deduce that whenever the agent has directly traveled between two distinctive states, the places associated with these distinctive states are different:

$$\vec{travel}(ds, ds') \wedge at(ds, p) \wedge at(ds', q) \rightarrow p \neq q$$

The statement above can be rewritten as

Corollary 1

$$\vec{travel}(ds, ds') \rightarrow place(ds) \neq place(ds')$$

where $\mathbf{place}(\mathbf{ds})$ denotes the unique topological place that distinctive state ds is at (axiom 5.16).

Moreover, consecutive travels among distinctive states occur along the same topological path, as stated in the next corollary.

Corollary 2

$$\bigwedge_{i=1}^{n-1} \vec{travel}(ds_i, ds_{i+1}) \rightarrow$$
$$\exists! pa, dir \quad \left[tpath(pa) \wedge \bigwedge_{i=1}^{n} \{order(pa, dir, place(ds_i), place(ds_{i+1})) \wedge along(ds_i, pa, dir)\} \right]$$

While the sorts of $places$ and $paths$ for a model M of $TT(E)$ are infinite, the interpretation of $tpath$ and $tplace$ is finite. Indeed,

Theorem 4 *The topological map associated with a finite set of experiences E has a finite number of topological paths and a finite number of topological places.*

Proof. See appendix B, page 216. □

Finally, it is important to note that our circumscription policy 5.37 and the fact that the sort of paths and places is infinite implies the following fact:[13]

Theorem 5 *Any two models of the SSH topological theory have the same number of topological paths and the same number of topological places.*

Proof. See appendix B (page 216). □

However, theorem 5 does not mean that a unique map is associated with a set of schemas. As shown in example 19 (page 59) the SSH topological theory could have more

[13]Recall that the interpretations for $tpath$ and $tplace$ are finite.

than one non-isomorphic model.

In order to prove that distinctive states ds_1 and ds_2 are at different topological places, one has to prove that $\neg turn\_eq(ds_1, ds_2)$. The following theorem states a strong condition for when this is case.

Theorem 6 *Let ds_1 be a distinctive state symbol such that*[14]

$$\forall ds_2 \notin [ds_1]_{\widehat{turn}},\ [ds_2]_{teq} \cap [ds_1]_{\widehat{turn}} = \emptyset \ .$$

Then

$$\forall ds_2 \notin [ds_1]_{\widehat{turn}},\ place(ds_2) \neq place(ds_1) \ .$$

Proof. The hypothesis of the theorem implies $\forall ds_2 \notin [ds_1]_{\widehat{turn}},\ \neg turn\_eq(ds_2, ds_1)$ (see appendix B, page 217). □

$\{$*end of remark*$\}$

The next examples illustrate the interplay among the axioms in AT_block.

Example 8

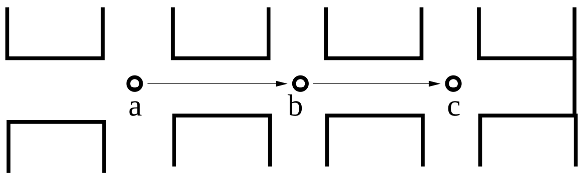


Figure 5.3: Distinctive states *a, b* and *c* cannot be distinguished at the causal level (see example 4, page 29). Using the concepts of paths and places these distinctive states become distinguishable.

Consider the set of experiences *E* as in Example 4 (page 29), Figure 5.3. From axiom 5.16 we conclude that there exist places P, Q, and R, such that $at(a, P), at(b, Q)$ and $at(c, R)$. Since it is the case that $\overrightarrow{travel}(a, b)$, from corollary 1 we conclude that $P \neq Q$. Similar arguments allow us to conclude that $Q \neq R$ and $P \neq R$. Consequently, the topological map associated with *E* has three topological places. Distinctive states a and c can be distinguished though they are "causally indistinguishable" (i.e. $ceq(a, c) \wedge \neg teq(a, c)$).

[14]*Given an equivalence relation R, $[x]_R$ denotes the equivalence class of x according to R.*

From corollary 2 we deduce that there exist a topological path Pa and direction Dir such that $order(Pa, Dir, P, Q)$ and $order(Pa, Dir, Q, R)$. Consequently, the topological map has one topological path and three topological places.
{*end of example*}

Only distinctive states linked by turn actions can be grouped into a topological place (axiom 5.17). Under incomplete information this constraint could imply the existence of more places than the ones needed.

Example 9

Consider the set of experiences E indicated by the formulae: $\langle\, a, travel, b\,\rangle$, $\langle\, b, turnAround, c\,\rangle$, $\langle\, c, travel, d\,\rangle$, in addition to the views associated with the distinctive states. Moreover, assume that views uniquely distinguish the different distinctive states. The model for *TT(E)* is presented in figure 5.4c. The model has three places and one path. Not having a $turn$ action relating d and a prevents the agent from grouping these distinctive states into the same place, as suggested in figure 5.4b. Next we show why this is the case.

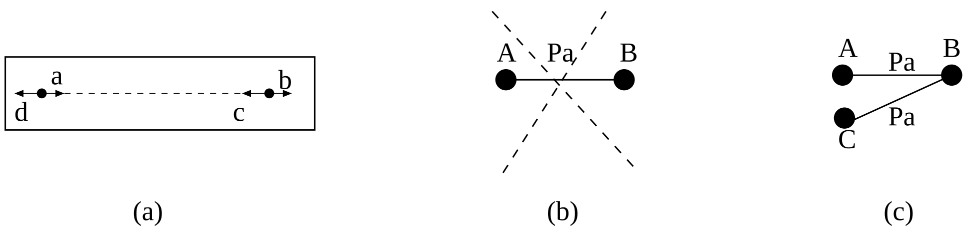


Figure 5.4: (a) Consider the set of experiences E given by $\langle\, a, travel, b\,\rangle$, $\langle\, b, turnAround, c\,\rangle$, $\langle\, c, travel, d\,\rangle$ corresponding to the agent navigating in a rectangle environment. The topological map associated with E, has three places and one path (c) rather than two places and one path (b). Distinctive states a and d cannot be grouped into the same topological place since they are not linked by turn actions. Should the agent turn around and experience the schema $\langle\, d, turnAround, a\,\rangle$, it will consider (b) as the topological map and disregard (c).

Since views uniquely distinguish distinctive states, then $teq(x, y) \equiv x = y$.[15] From the definition of $turn\_eq$ (block 5.18), it follows then that $turn\_eq = \widehat{turn}$. We can calculate $\widehat{turn}$ as follows. The only $turn$ action mentioned in E is the one in schema $\langle\, b, turnAround, c\,\rangle$. Consequently, from block 5.6 we deduce

[15]See section 5.4, page 60.

$\widehat{turn}(ds, ds') \equiv [ds = ds' \vee \{ds = b \wedge ds' = c\} \vee \{ds = c \wedge ds' = b\}]$. In particular, $\neg turn\_eq(a, d)$. In virtue of axiom 5.17 we cannot conclude that a and d are at the same topological place.

Notice that the model associated with E has three places and one path. The order of places in the path is not total. Should the agent turn around and experience the schema $\langle\, d, turnAround, a\,\rangle$, the new set of experiences will have a model with two places and one path as suggested in figure 5.4b.
{*end of example*}

The next example shows the interplay between teq and $along$ as well as the effect of maximizing teq.

Example 10

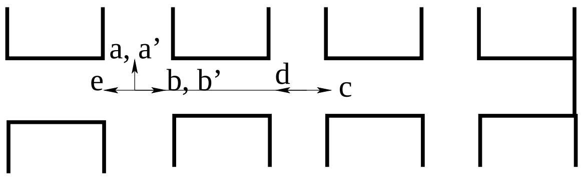


Figure 5.5: The agent moves back and forth from one intersection to the other. The second time the agent visits distinctive states a and b, it gives the names a' and b'. Our topological theory will conclude that these names correspond to the previously visited a and b.

Consider the set of schemas

$$\langle a, turnRight, b\rangle \qquad \langle b, travel, c\rangle \qquad \langle c, turnAround, d\rangle$$
$$\langle d, travel, e\rangle \quad \langle e, turnRight, a'\rangle \quad \langle a', turnRight, b'\rangle$$

consistent with an agent going from one four-way intersection to another (see figure 5.5). Let's consider the models of these schemas. From our axioms, at least one path and three places must exist:

We know that $P \neq Q$ and $Q \neq R$. By having $teq(a, a')$, we can complete the model such that $P = R$. The question rises of whether we can maximize teq by asserting $teq(b, b')$. If we did not include axiom 5.23 in the $along$ block (block 5.20), we could not make $teq(b, b')$ since there is not tuple of the form $along(b', \bullet, \bullet)$ (axiom 5.11). However, by including this axiom in the block, we are allowed to have a model in which $teq(b, b')$ is

Places	*Paths*	*Along*	teq
P = a,b	Pa: b-c d-e	along(b,Pa,dir) along(c,Pa,dir)	$\neg teq(a,b)$, $\neg teq(c,d)$
Q = c,d		along(d,Pa,-dir) along(e,Pa,-dir)	$\neg teq(e,a')$, $\neg teq(a',b')$
R = e,a',b'			

the case.[16] The maximization of teq will force the model to have $teq(b,b')$.
{*end of example*}

Example 11

Consider an extension of the previous example by adding the schema $\langle b', travel, c'\rangle$. We have the following situation:

Places	*Paths*	*Along*
P = {a,b}	Pa: b-c {d-e}	along(b,Pa,dir) along(c,Pa,dir)
Q = {c,d}		along(d,Pa,-dir) along(e,Pa,-dir)
R = {e,a',b'}		
S = {c'}	Pa1: b'-c'	along(b',Pa1,dir1) along(c',Pa1,dir1)

By making $teq(b,b')$ it is possible to have $Pa = Pa1$. Notice that in this case one does not need to consider axiom 5.23 inside the along block (block 5.20). This is the case since there are tuples of the form $along(b', \bullet, \bullet)$ implied by the schemas. By making $teq(c,c')$, one show that it is possible to have $Q = S$ and so a model with two places and one path. It remains to see that by maximizing teq, $teq(a,a')$ will be the case.[17]
{*end of example*}

The next example illustrates how sometimes the maximization of teq might give unexpected results.

Example 12

Consider the schemas $\{\langle a, turnRight, b\rangle, \langle b, turnRight, c\rangle\}$ where a, b and c have the same view. From axiom 5.12, $\neg teq(a,b)$ and $\neg teq(b,c)$ are the case. Maximizing teq will

[16]The problem is that a travel action has not been performed at b' and so the schemas do not support a tuple of the form $along(b', \bullet, \bullet)$.

[17]Without maximizing teq nothing forces us to identify a and a'.

imply $teq(a, c)$.

One could argue that the maximization is doing the right thing since there is not much one can deduce from $turnRight$ schemas alone. Either extra information (more schemas) or metrical information is needed to distinguish a from c (if indeed they are distinguishable).
{*end of example*}

By requiring the agent to have traveled among the places on a same path (axiom 5.33), different paths can be identified. The next example illustrates the case.

Example 13

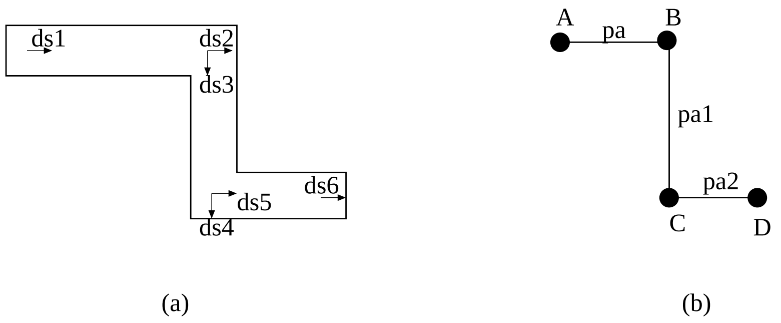


Figure 5.6: By requiring the agent to have traveled among the places on a same path (axiom 5.33), different paths can be identified. (a) The agent visits the different distinctive states in the order $ds1, ds2, \ldots, ds6$. (b) depicts the topological map associated with (a). Three paths instead of only two are required to explain the agent experiences (see text).

Suppose the agent explores the environment depicted in figure 5.6a obtaining the following schemas:

$$\begin{array}{ll} \langle ds1, travel, ds2 \rangle & \\ \langle ds2, turnRight, ds3 \rangle & \langle ds3, travel, ds4 \rangle \\ \langle ds4, turnLeft, ds5 \rangle & \langle ds5, travel, ds6 \rangle \end{array}$$

We assume that the agent associates different views with the different distinctive states in the example. Axiom 5.16 implies that there exist places A, B, C and D (see figure 5.6b) such that

$$at(ds1, A), \;\; at(ds2, B), \;\; at(ds3, B), \;\; at(ds4, C), \;\; at(ds5, C), \;\; at(ds6, D) \;\; .$$

Moreover, corollary 1 implies that

$$A \neq B \ , \ B \neq C \ , \ C \neq D \ \ .$$

Under our assumption that all distinctive states in the example have different views, it follows then that $teq(ds_1, ds_2) \equiv ds_1 = ds_2$ and thus $\widehat{turn} = turn\_eq$ (see section 5.4). Since $\neg\widehat{turn}(ds1, ds3)$, $\neg\widehat{turn}(ds1, d5)$ and $\neg\widehat{turn}(ds2, ds6)$ are the case, it follows that

$$A \neq C, \ \ A \neq D, \ \ B \neq D \ \ .$$

Consequently, places A, B, C and D are all different.

Axiom 5.21 implies that there exist paths Pa, $Pa1$, $Pa2$, and directions dir, $dir1$, $dir2$, such that:

$$order(Pa, dir, A, B), \ \ along(ds1, Pa, dir), \ \ along(ds2, Pa, dir),$$

$$order(Pa1, dir1, B, C), \ \ along(ds3, Pa1, dir1), \ \ along(ds4, Pa1, dir1),$$

$$order(Pa2, dir2, C, D), \ \ along(ds5, Pa2, dir2), \ \ along(ds6, Pa2, dir2) \ \ .$$

Since we have the schemas $\langle ds2, turnRight, ds3\rangle$ and $\langle ds4, turnLeft, ds5\rangle$, axiom 5.26 implies that

$$Pa \neq Pa1, \ \ Pa1 \neq Pa2 \ \ .$$

Since $teq(ds_1, ds_2) \equiv ds_1 = ds_2$ and there is not $turnAround$ schemas in E, then $\widehat{travel} = travel\_eq$. Consequently $\neg\widehat{travel}(ds1, ds4)$ and $\neg\widehat{travel}(ds1, ds5)$ are the case, and in virtue of axiom 5.33 it follows that

$$Pa \neq Pa2 \ \ .$$

{*end of example*}

Next we illustrate how the predicate $travel\_eq$ is used when defining topological paths.

Example 14

Consider the environment in figure 5.7. Suppose the agent starts at distinctive state a and visits places A,B,C,D,E, and F as suggested by the arrows in the figure.[18] The corresponding set of experiences contains the formulae $View(a, v_a)$, $View(b, v_b)$, $\langle a, travel, b\rangle$, etc..

[18]A capital letter L denotes the topological place associated with distinctive state l.

In addition, suppose that the views associated with the explored distinctive states are all different. Once at F, the agent travels to A (though the agent does not know it), and adds to the set of experiences the formulae $Views(f, v_f)$, $View(a', v_a)$, $\langle f, travel, a' \rangle$, where a' is a distinctive state name not used before. Notice that we use a' instead of a since the agent does not know that it is back to a. The agent concludes that is back to a' by deducing that $teq(a, a')$. Next we explain why this will be the case.

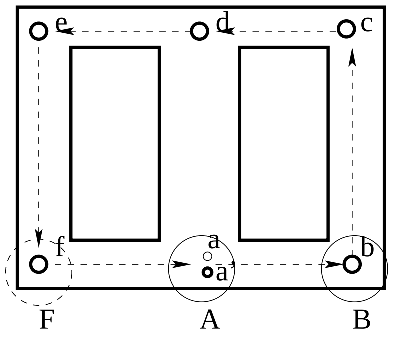


Figure 5.7: By identifying a and a' one path containing places F, A, and B is created. Notice that in this case, $\neg\widehat{travel}(f, b)$ and $travel\_eq(f, b)$ hold. (see text)

We assume that in this environment the only distinctive states with view v_a are a and a'. Since no action has been executed at a', we trivially satisfy axiom 5.10 when declaring $teq(a, a')$. By declaring $at(a', A)$, we can have one topological path Pa such that $order(Pa, pos, F, A)$ and $order(Pa, pos, A, B)$. The model obtained this way will be minimal since at least four paths are needed in a topological map of figure 5.7.

Notice that $\neg\widehat{travel}(f, b)$ holds. However, it is the case that $travel\_eq(f, b)$, since a and a' are topologically indistinguishable (i.e. $teq(a, a')$) (see block 5.34).
{*end of example*}

Example 15

Consider the same T-environment exploration presented in example 6 (page 30) (see figure 5.8). When using only causal information, three possible models are associated with the exploration. When using topological information, only one of these models is possible as illustrated next.

The three causal models associated with T-environment are:

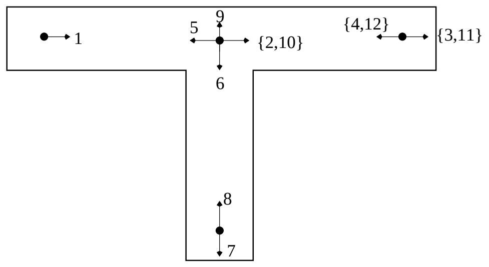


Figure 5.8: The agent visits the different distinctive states in the order suggested by their numbers. The same travel action ml is performed when traveling from a corner to the intersection (i.e $\langle\, 1, ml, 2\,\rangle$) and viceversa (i.e. $\langle\, 4, ml, 5\,\rangle$). A turn around action is performed when reaching a corner (i.e. $\langle\, 3, turnAround, 4\,\rangle$,$\langle\, 7, turnAround, 8\,\rangle$, etc.). Assume that the different corners have the same views (i.e. view(1) = view(4) = view(8), view(3)= view(7) = view(11)), and views associated with the other distinctive states are different. Three different causal models can be associated with the agent exploration of this T-environment but only one of them is consistent with topological information (see text).

1. Model 1: $ceq(8, 12), ceq(12, 8), ceq(x, x)$.[19]
2. Model 2: $ceq(1, 12), ceq(12, 1), ceq(x, x)$.
3. Model 3: $ceq(4, 12), ceq(12, 4), ceq(3, 11), ceq(11, 3), ceq(2, 10), ceq(10, 2), ceq(x, x)$.

We are to show that only model 3 above is consistent with topological information. For this we show the following three facts: (i) any model must have at least 2 tpaths and 5 tplaces,[20] (ii) there is a model with 2 tpaths and 5 tplaces (this is the intended model), (iii) a model of $\neg teq(2, 10)$ must have at least 6 tplaces. This last statement implies that models 1 and 2 above are not consistent with topological information.

From $\langle\, 1, travel, 2\,\rangle$ and $\langle\, 2, travel, 3\,\rangle$, corollary 2 implies that there exist a path $Pa1$ and direction $dir1$ such that

$$along(1, Pa1, dir1),\ along(2, Pa1, dir1),\ along(3, Pa1, dir1)\ \ .$$

Moreover, corollary 1 implies that

$$place(1) \neq place(2),\ place(2) \neq place(3),\ place(1) \neq place(3)\ \ .$$

From $\langle\, 3, turnAround, 4\,\rangle$, $\langle\, 4, travel, 5\,\rangle$, axiom 5.22 and corollary 2, it is the case that

$$along(4, Pa1, -dir1),\ along(5, Pa1, -dir1)\ .$$

[19]The extent of ceq in model 1 is defined by $\{(8, 12), (12, 8)\} \cup \{(x, x)\ : x = 1, \ldots, 12\}$.

[20]Since there is not a turn action between dstates $\{5, 6\}$ and dstates $\{2, 9, 10\}$, these dstates are not at the same topological place, as suggested by figure 5.8.

Similarly, from $\langle\, 5, turnLeft, 6\,\rangle$, $\langle\, 6, travel, 7\,\rangle$, $\langle\, 7, turnAround, 8\,\rangle$, $\langle\, 8, travel, 9\,\rangle$ we conclude that there exist a path $Pa2$ and direction $dir2$ such that $Pa1 \neq Pa2$ (axiom 5.26) and

$$place(5) \neq place(8),\ along(6, Pa2, dir2),\ along(7, Pa2, dir2),$$

$$along(8, Pa2, -dir2),\ along(9, Pa2, -dir2)\ \ .$$

From $\langle\, 9, turnRight, 10\,\rangle$, $\langle\, 10, travel, 11\,\rangle$, $\langle\, 11, turnAround, 12\,\rangle$, there exist path $Pa3$ and direction $dir3$ such that $Pa2 \neq Pa3$ and

$$along(10, Pa3, dir3), along(11, Pa3, dir3), along(12, Pa3, -dir3)\ .$$

Theorem 6 allow us to conclude that $place(5) \notin \{place(1), place(2), place(3)\}$. The same argument shows that $place(8) \notin \{place(1), place(2), place(3), place(5)\}$. *Consequently, a miminal model of the theory must have at least two tpaths and five tplaces.*

Notice that in the intended model of the T-environment, $Pa1 = Pa3$, $dir1 = dir3$, $teq(2, 10)$, $teq(3, 11)$ and $teq(4, 12)$. This model is indeed a model of $TT(E)$ since at least two topological paths and five topological places are needed to explain E, and consequently any model must have two topological paths and five topological places (theorem 5).

If $\neg teq(2, 10)$ were the case, then theorem 6 allows to conclude that $place(9) \notin \{place(1), place(2), place(3), place(5), place(8)\}$ and so the model will have at least six tplaces. Consequently $teq(2, 10)$ has to be the case in a minimal model of the theory.
$\{$*end of example*$\}$

Example 16

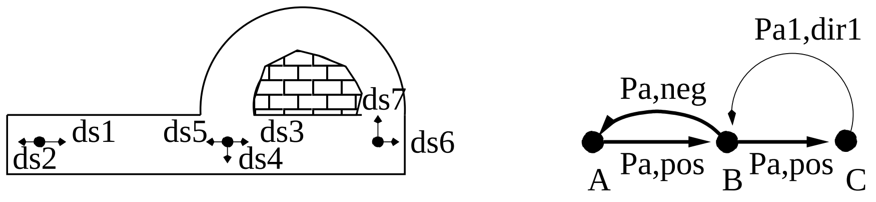
Consider an extension of the previous example where we have the additional schemas

$$\langle\, 9, turnLeft, 5'\,\rangle,\ \langle\, 5', turnRight, 9\,\rangle\ .$$

In this case, the intended model has *four* places and two paths. Notice that now the agent can conclude that $place(5) = place(2)$ by making $teq(5', 5)$ and so $turn\_eq(5, 2)$. $\{$*end of example*$\}$

Our theory does not assume that paths intersect at most in one place. Consider the next example.

Example 17



(a) (b)

Figure 5.9: The environment in (a) illustrates a case where different paths intersect at more than one place. Suppose the agent explores the environment by visiting the different distinctive states in the order $ds1, ds2, ds1, ds3, ds4, ds3, ds6, ds7, ds4, ds5, ds2$. (b) depicts the topological map associated with this environment.

Suppose the agent explores the environment depicted in figure 5.9 obtaining the following schemas:

$$\begin{array}{lll}
\langle ds1, turnAround, ds2\rangle & \langle ds2, turnAround, ds1\rangle & \langle ds1, travel, ds3\rangle \\
\langle ds3, turnRight, ds4\rangle & \langle ds4, turnLeft, ds3\rangle & \langle ds3, travel, ds6\rangle \\
\langle ds6, turnLeft, ds7\rangle & \langle ds7, travel, ds4\rangle & \\
\langle ds4, turnRight, ds5\rangle & \langle ds5, travel, ds2\rangle &
\end{array}$$

We assume that views (which we omit) uniquely distinguish the different distinctive states. From corollary 1 there exist the different places A,B, and C suggested in the figure. In addition, corollary 2 implies the existence of a path, Pa, and direction, say pos, such that

$$order(Pa, pos, A, B) \ \ , \ \ order(Pa, pos, B, C) \ \ order(Pa, pos, A, C) \ \ .$$

Moreover, from schemas $\{\langle ds7, travel, ds4\rangle, \langle ds5, travel, ds2\rangle\}$ and axiom 5.21, there exist paths $Pa1$, $Pa2$, and directions $dir1$, $dir2$, such that

$$order(Pa1, dir1, C, B) \wedge along(ds7, Pa1, dir1) \wedge along(ds4, Pa1, dir1) \ ,$$

$$order(Pa2, dir2, B, A) \wedge along(ds5, Pa2, dir2) \wedge along(ds2, Pa2, dir2) \ \ .$$

Since $along(ds6, Pa, pos)$, from axiom 5.26 and schema $\langle ds6,\ turnLeft,\ ds7\rangle$ we conclude that

$$Pa \neq Pa1 \ \ .$$

Since we are minimizing paths, by setting $Pa2 = Pa \wedge dir2 = neg$, we obtain a minimal model for E. Notice that in this model, places B and C belong to two different paths, Pa and $Pa1$.

{*end of example*}

Remark. As the example above illustrates, in our theory it is the case that:

$$order(pa, dir, p, q) \not\to \exists ds \ \{at(ds, p) \wedge along(ds, pa, dir)\} \tag{5.38}$$

$$order(pa, dir, p, q) \wedge order(pa1, dir1, p, q) \not\to \{pa = pa1 \wedge dir = dir1\} \tag{5.39}$$

Formula 5.38 expresses the fact that the order of places along a given path's direction can be stated even though it cannot be stated what direction (distinctive state) to take in order to follow the path. This does not come as a surprise given axiom 5.29 which relates the order of places on both directions of a given path.[21] Moreover, the block defining $along$ (block 5.20, page 41) embodies the default $\neg along(ds, pa, dir)$.

Formula 5.39 comes as a surprise. It states that different paths can have the same order of places. In particular, from the order of places one cannot derive the identity of the path. This is the case since we have not assumed a "rectilinear" environment where paths intersect at most in one place. Our circumscription policy however, embodies the "default"

$$order(pa, dir, p, q) \wedge order(pa1, dir1, p, q) \to \{pa = pa1 \wedge dir = dir1\} \quad .$$

{*end of remark*}

There are some patterns of experience in which our theory is not applicable. In particular, axiom 5.24 rules out experiences where "known" different paths merge into the same distinctive state. The following example illustrates the case.

Example 18

Consider the environment depicted in figure 5.10. Suppose the agent has experienced the following schemas:

$$\begin{array}{ll} \langle b, travel, d\rangle & \langle d, turnAround, c\rangle \\ \langle c, turnRight, e\rangle & \langle e, travel, a\rangle \\ \langle a, turnAround, b\rangle & \end{array}$$

From axiom 5.21 we know that exist paths Pa, $Pa1$ and directions dir, $dir1$ such that

$$along(b, Pa, dir) \wedge along(d, Pa, dir) \ ,$$

[21]For a simple example of formula 5.38, consider the topological map associated with the schema $\langle a, travel, b\rangle$.

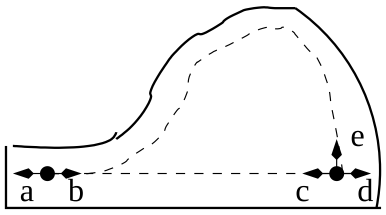


Figure 5.10: Distinctive state a is along two different paths, which contradicts axiom 5.24. Experiences like the one in the figure are handled by the SSH general topological theory described in section 5.5.

$$along(e, Pa1, dir1) \wedge along(a, Pa1, dir1) \ .$$

Moreover, from axiom 5.22 it follows that

$$along(b, Pa1, -dir1) \ .$$

From axiom 5.24 we conclude that $Pa = Pa1$. However, from schema $\langle c, turnRight, e\rangle$ and axiom 5.26 we conclude that $Pa \neq Pa1$ which is a contradiction.
$\{$*end of example*$\}$

The previous example illustrates plausible patterns of experience that realistic environments could generate. In section 5.5 we extend our theory to cope with convergent paths like the ones in the previous example.

5.2 teq versus ceq

It is worth noticing that teq satisfies the same axioms inside the CEQ_block.[22] Consequently, given an interpretation for teq it could be extended to at least one interpretation for ceq. However, the converse it not true. Given an interpretation for ceq it does not necessarily have a subset that defines an interpretation for teq. For instance, in example 15 (page 53) only one of the three possible interpretations for ceq admits a topological structure.

It is tempting to include an axiom like $teq(ds_1, ds_2) \rightarrow ceq(ds_1, ds_2)$ in the definition of teq (the AT_block 5.7, page 41). However this will yield the wrong results since ceq is not varied in the block defining teq. In particular, theorem 5 will not be the case (will not be a theorem!!), and model 2 in example 15 will admit a topological map with two tpaths and six tplaces.

[22]Compare axioms 5.9-5.10 (page 41) in the definition of teq with axiom 4.24-4.25 in ceq's definition (page 26.

5.3 The SSH topological map

Given a minimal model M of $TT(E)$, the SSH topological map is defined by the extent in M of *tpath*, *tplace*, *along*, *order*, *on*, *at* and *teq*. Notice that axioms 5.15 and 5.19 restrict the domain of at and $along$ to topological places and topological paths. The domain of $order$ is implicitly restricted to topological paths in virtue of block 5.27 and axiom 5.21.

While the sorts of $places$ and $paths$ for a model M of $TT(E)$ are infinite, the interpretation of $tpath$ and $tplaces$ is finite. Indeed,

Theorem 4 *The topological map associated with a finite set of experiences E has a finite number of topological paths and a finite number of topological places.*

Proof. See appendix B, page 216. □

Since the positive and negative direction of a path are chosen arbitrarily (Axiom 5.21), there is not a unique minimal model for $TT(E)$. Given any model M of $TT(E)$ one could define another model M' of $TT(E)$ by choosing a path pa in M and reversing the roles of the directions pos and neg for pa:

- $order^{M'}(Pa, pos, A, B)$ whenever $order^{M}(Pa, neg, A, B)$.[23]
- $order^{M'}(Pa, neg, A, B)$ whenever $order^{M}(Pa, pos, A, B)$.
- $along^{M'}(ds, Pa, pos)$ whenever $along^{M}(ds, Pa, neg)$.
- $along^{M'}(ds, Pa, neg)$ whenever $along^{M}(ds, Pa, pos)$.
- M' and M are equal otherwise.

We will consider these "up to path direction isomorphic" models to be the same. However, it is still the case that the theory $TT(E)$ has minimal models that are not isomorphic up to path direction. In general, when distinctive states look alike (i.e. have the same view), the agent's experiences might not be enough to distinguish them. The next example illustrates this case.

Example 19

[23] $order^{M'}(Pa, pos, A, B)$ stands for the tuple (Pa, pos, A, B) belonging to the interpretation of $order$ in the model M'.

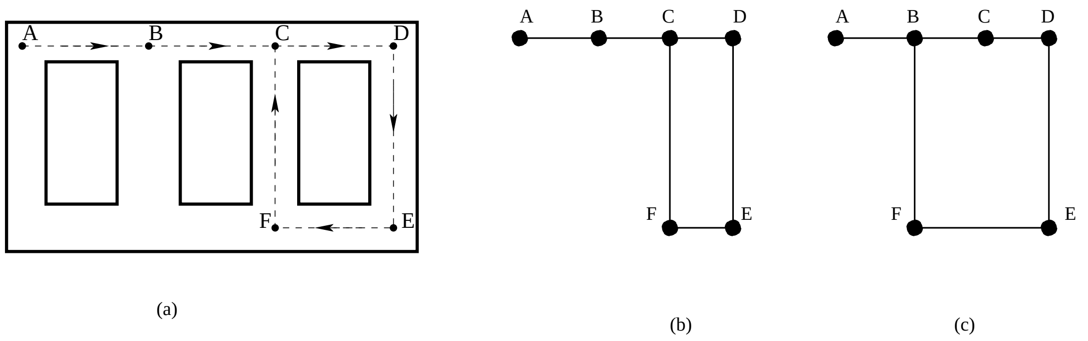


Figure 5.11: (a) The agent goes around the block visiting places A,B,...,F,C in the order suggested in the figure. Intersections B and C look alike to the agent. (b) and (c) represent two possible representations for the environment in (a). Topological information is not enough to decide whether the agent is back to B or C.

Assume that the agent visits places A,B,C,D,E,F,C in the order suggested in figure 5.11. We do not include the corresponding set of schemas, but we assume that intersections look alike, and turn actions were executed at places D, E and F. As view information is not enough to distinguish intersections, places B and C look alike. Given this information, the agent is not able to decide whether it is back to B or C and consequently two minimal models can be associated with the set of experiences in this environment (figures 5.11b,c).

Notice that if the agent accumulates more information, by turning at C and traveling to D, then it can deduce that the topology of the environment is the one in figure 5.11b. This is the case since the views at C and D are different.[24]
{*end of example*}

5.4 What if Views uniquely identify distinctive states

Our theory explicitly handles "perceptual aliasing" by introducing the predicate teq. This predicate plays the role of "equality", and so we explicitly have to handle the "replacement" properties teq satisfies (see axioms 5.18, 5.23, 5.34).

It is possible to somewhat simplify our theory when views uniquely identify distinctive states. Under this hypothesis, we replace axiom 4.13 (page 22) by the following

[24]Metrical information can be used to deduce the correct topology. See example 25 (page 100).

axiom:

$$View(ds_1, V) \wedge View(ds_2, V) \rightarrow ds_1 = ds_2 \tag{5.40}$$

In virtue of axioms 4.24 and 5.9, $ceq \subseteq =$ and $teq \subseteq =$, and being ceq and teq equivalence relations we conclude that they can be replaced by equality. The fact that ceq and teq reduce to $=$ is expected since all that is required to identify a distinctive state is its view.

By replacing teq by $=$, block 5.18 can be rewritten as:

$$\begin{array}{ll}
\{ & min\ \ turn\_eq : \\
 & \widehat{turn}(ds_2, ds_3) \wedge teq(ds_1, ds_2) \wedge teq(ds_3, ds_4) \rightarrow turn\_eq(ds_1, ds_4) \\
 & turn\_eq(ds_1, ds_2) \wedge turn\_eq(ds_2, ds_3) \rightarrow turn\_eq(ds_1, ds_3) \\
\} & \\
\quad \equiv & \\
\{ & min\ \ turn\_eq : \\
 & \widehat{turn}(ds_2, ds_3) \wedge ds_1 = ds_2 \wedge ds_3 = ds_4 \rightarrow turn\_eq(ds_1, ds_4) \\
 & turn\_eq(ds_1, ds_2) \wedge turn\_eq(ds_2, ds_3) \rightarrow turn\_eq(ds_1, ds_3) \\
\} & \\
\quad \equiv & \\
\{ & min\ \ turn\_eq : \\
 & \widehat{turn}(ds_1, ds_2) \rightarrow turn\_eq(ds_1, ds_2) \\
 & turn\_eq(ds_1, ds_2) \wedge turn\_eq(ds_2, ds_3) \rightarrow turn\_eq(ds_1, ds_3) \\
\} & \\
\quad \equiv & \\
 & \widehat{turn} = turn\_eq
\end{array}$$

where the last equality follows from the fact that $\widehat{turn}$ is transitive, and $turn\_eq$ is the minimum transitive predicate containing $\widehat{turn}$. Consequently, we do not need the predicate $turn\_eq$ in our formalization.

As for $travel\_eq$ we can do some simplifications:

$$\begin{array}{l}
\{\, min\ \ travel\_eq : \\
\quad \widehat{travel}(ds_1, ds_2) \rightarrow travel\_eq(ds_1, ds_2), \\
\quad \langle\, ds_1, TurnAround, ds_2 \,\rangle \rightarrow travel\_eq(ds_1, ds_2) \wedge travel\_eq(ds_2, ds_1) \\
\quad teq(ds_1, ds_2) \wedge teq(ds_3, ds_4) \wedge travel\_eq(ds_2, ds_3) \rightarrow travel\_eq(ds_1, ds_4), \\
\quad travel\_eq(ds_1, ds_2) \wedge travel\_eq(ds_2, ds_3) \rightarrow travel\_eq(ds_1, ds_3)
\end{array}$$

$$
\begin{array}{l}
\}\\
\equiv\\
\{\ min\ \ travel\_eq :\\
\quad \widehat{travel}(ds_1, ds_2) \rightarrow travel\_eq(ds_1, ds_2),\\
\quad \langle\, ds_1, TurnAround, ds_2\, \rangle \rightarrow travel\_eq(ds_1, ds_2) \wedge travel\_eq(ds_2, ds_1)\\
\quad travel\_eq(ds_1, ds_2) \wedge travel\_eq(ds_2, ds_3) \rightarrow travel\_eq(ds_1, ds_3)\\
\}
\end{array}
$$

Therefore, if views uniquely identify distinctive states, **AT_block** can be rewritten as follows:

$$
\begin{array}{l}
AT\_block =\\
\quad \{ :\\
\qquad \langle ds, turn, ds'\rangle \rightarrow ds \neq ds',\\
\qquad \langle ds, turnAround, ds'\rangle \wedge \langle ds, TurnAround, ds''\rangle \rightarrow ds' = ds'',\\
\qquad \langle\, ds_1, turnAround, ds_2\, \rangle \wedge \langle\, ds_2, turnAround, ds_3\, \rangle \rightarrow ds_1 = ds_3\ ,\\
\\
\qquad at(ds, p) \rightarrow tplace(p),\\
\qquad \exists! p\, at(ds, p),\\
\qquad \widehat{turn}(ds_1, ds_2) \equiv \forall p\, [at(ds_1, p) \equiv at(ds_2, p)]\, ,\\
\\
\qquad along(ds, pa, dir) \rightarrow tpath(pa),\\
\qquad \{\ min\ along :\\
\qquad\quad \langle\, ds, travel, ds'\, \rangle \rightarrow \exists pa,\ dir\ [along(ds, pa, dir) \wedge along(ds', pa, dir)]\, ,\\
\qquad\quad \langle ds, turnAround, ds'\rangle \rightarrow along(ds, pa, dir) \equiv along(ds', pa, -dir),\\
\qquad \}\\
\qquad along(ds, pa, dir) \wedge along(ds, pa1, dir1) \rightarrow pa = pa1 \wedge dir = dir1,\\
\qquad at(ds_1, p) \wedge at(ds_2, p) \wedge along(ds_1, pa, dir) \wedge along(ds_2, pa, dir) \rightarrow ds_1 = ds_2,\\
\\
\qquad [\langle ds, turn\_desc, ds'\rangle \wedge turn\_desc \neq turnAround \wedge\\
\qquad\quad along(ds, pa, dir) \wedge along(ds', pa1, dir1)] \rightarrow pa \neq pa1,\\
\\
\qquad \{\ min\ order :\\
\qquad\quad [\langle\, ds, travel, ds'\, \rangle \wedge at(ds, p) \wedge at(ds', q) \wedge\\
\qquad\qquad along(ds, pa, dir) \wedge along(ds', pa, dir)] \rightarrow order(pa, dir, p, q),\\
\qquad\quad order(pa, pos, p, q) \equiv order(pa, neg, q, p),
\end{array}
$$

$$
\begin{array}{l}
\quad order(pa, dir, p, q) \wedge order(pa, dir, q, r) \rightarrow order(pa, dir, p, r) \\
\} \\
\neg order(pa, dir, p, p), \\
\\
\{min\ on:\ at(ds, p) \wedge along(ds, pa, dir) \rightarrow on(pa, p)\ \} \\
on(pa, p) \wedge on(pa, q) \wedge tpath(pa) \rightarrow \\
\qquad \exists ds_1, dir_1, ds_2, dir_2\ [at(ds_1, p) \wedge along(ds_1, pa, dir_1) \wedge at(ds_2, q) \wedge \\
\qquad\qquad\qquad\qquad along(ds_2, pa, dir_2) \wedge travel\_eq(ds_1, ds_2)]\,, \\
\{min\ \ travel\_eq: \\
\quad \widehat{travel}(ds_1, ds_2) \rightarrow travel\_eq(ds_1, ds_2), \\
\quad \langle\, ds_1, turnAround, ds_2\, \rangle \rightarrow travel\_eq(ds_1, ds_2) \wedge travel\_eq(ds_2, ds_1) \\
\quad travel\_eq(ds_1, ds_2) \wedge travel\_eq(ds_2, ds_3) \rightarrow travel\_eq(ds_1, ds_3) \\
\} \\
\\
\textbf{circ}\ \ tpath \succ tplace\ \ \textbf{var}\ \ at,\ along,\ order,\ on,\ travel\_eq
\end{array}
$$

}

5.5 Coping with self intersecting paths

The topological theory presented in the previous sections is adequate for representing environments where "complex" paths configurations do not occur. In particular, we assume that self-intersecting and convergent paths do not exist. In this section we extend our theory to deal with these types of paths (see figure 5.12). Convergent paths are the standard counterexample for our axiom stating that distinctive states are along a unique path (axiom 5.24). Self-intersecting paths are the standard counterexample for the axioms stating that: (a) turning changes the path (axiom 5.26), (b) at a place there is at most one distinctive state along a path direction (axiom 5.25), and (c) the order of places in a path is not reflexive (axiom 5.31). We will make the axioms above defeasible statements, and embody in our theory the default that "normally, convergent or self-intersecting paths do not exists". Moreover, our changes are such that any map derived with the old theory is still a map for the new one.

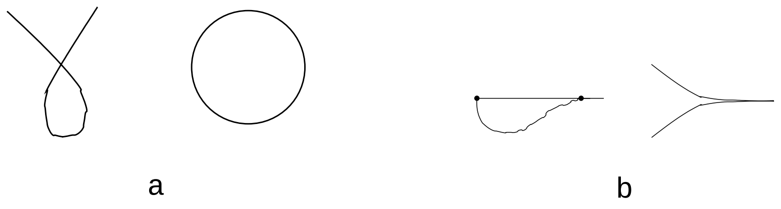


Figure 5.12: (a) Self intersecting paths. (b) Convergent paths.

5.5.1 Converging paths

Converging paths are the standard counterexample for the axiom stating that distinctive states are along a unique path (axiom 5.24). We replace

$$along(ds, pa, dir) \wedge along(ds, pa1, dir1) \rightarrow pa = pa1 \wedge dir = dir1,$$

by the block

$$\begin{aligned} &\{ \; min\ convergent\_paths : \\ &\quad [along(ds, pa, dir) \wedge along(ds, pa1, dir1) \\ &\qquad \wedge \neg [pa = pa1 \wedge dir = dir1]] \rightarrow convergent\_paths(pa, pa1) \\ &\} \end{aligned}$$

The block defining *convergent_paths* does not specify that we prefer models where such paths do not exist (see section 5.5.4, page 68). For this, we need to include *convergent_paths* in our circumscription policy (see section 5.5.3).

5.5.2 Self-intersecting paths

Self-intersecting paths are the standard counterexample for the axioms stating that turning changes the path (axiom 5.26), at a place there is at most one distinctive state along a path direction (axiom 5.25), and the order of places in a path is not reflexive (axiom 5.31). We replace

$$\begin{aligned} &\neg order(pa, dir, p, p), \\ &[\langle ds, turn\_desc, ds' \rangle \;\wedge\; turn\_desc \neq turnAround \wedge \\ &\qquad along(ds, pa, dir) \wedge along(ds', pa1, dir1)] \rightarrow pa \neq pa1, \\ &at(ds_1, p) \wedge at(ds_2, p) \wedge along(ds_1, pa, dir) \wedge along(ds_2, pa, dir) \rightarrow teq(ds_1, ds_2), \end{aligned}$$

by the block

$$\begin{aligned} &\{min\ self\_intersecting\ : \\ &\quad order(pa, dir, p, p) \rightarrow self\_intersecting(pa)\ , \end{aligned}$$

$$[\langle ds, turn\_desc, ds'\rangle \wedge turn\_desc \neq TurnAround \wedge along(ds, pa, dir)$$
$$\wedge along(ds', pa, dir1)] \rightarrow self\_intersecting(pa)\ ,$$
$$[at(ds_1, p) \wedge at(ds_2, p) \wedge along(ds_1, pa, dir) \wedge along(ds_2, pa, dir)$$
$$\wedge \neg teq(ds_1, ds_2)] \rightarrow self\_intersecting(pa)$$
$$\}$$

Like convergent paths, the block defining $self\_intersecting$ does not specify that we prefer models where such paths do not exist (see Section 5.5.4, page 68). For this, we need to include $self\_intersecting$ in our circumscription policy (see section 5.5.3).

5.5.3 New circumscription policy

While we have defined convergent and self-intersecting paths, we still need to formalize that by default these kind of paths do not exist. This is accomplished by giving priority to the minimization of these two predicates over any other predicate. The new circumscription policy associated with our theory becomes

$$\textbf{circ}\ self\_intersecting \succ convergent\_paths \succ tpath \succ tplace\ \textbf{var}\ \vec{SSH}pred. \quad (5.41)$$

The intuition behind the predicate's order in the circumscription is as follows: clearly axioms 5.26 and 5.25 are about distinguishing paths, and so we need to minimize $self\_intersecting$ even at the expense of having more topological paths.[25] Example 20 illustrates why we give priority to $self\_intersecting$ over $convergent\_paths$. As for the order $convergent\_paths \succ tpath \succ tplace$, we want to guarantee that maps according to the old theory are still maps according to the new theory. In this sense, the new theory is a "conservative" extension of our previous theory, as stated in the following theorem:

Theorem 7 *Any topological map with respect to our previous theory is a topological map according to the new theory.*

[25]Notice that axiom 5.25 can be rewritten as follows:

$$\begin{array}{ll} & at(ds_1, p) \wedge at(ds_2, p) \wedge along(ds_1, pa, dir) \wedge along(ds_2, pa, dir) \rightarrow teq(ds_1, ds_2) \\ \equiv & at(ds_1, p) \wedge at(ds_2, p) \wedge along(ds_1, pa, dir) \wedge along(ds_2, pa1, dir) \wedge pa = pa1 \rightarrow teq(ds_1, ds_2) \\ \equiv & at(ds_1, p) \wedge at(ds_2, p) \wedge along(ds_1, pa, dir) \wedge along(ds_2, pa1, dir) \wedge \neg teq(ds_1, ds_2) \rightarrow pa \neq pa1 \end{array}$$

We beg the issue that axiom 5.31 is more about distinguishing places than it is about distinguishing paths. In fact

$$\neg order(pa, dir, p, p) \equiv \neg\,[order(pa, dir, p, q) \wedge p = q] \equiv [order(pa, dir, p, q) \rightarrow p \neq q]\ .$$

At the very least we need to have $self\_intersecting \succ tplace$ in the circumscription policy.

In particular, the maps associated with examples 8 through 17 are still valid maps for the new theory. Next we study some cases we could not handle before.

Example 20

Consider the same scenario of example 18 (see figure 5.13, below). Suppose the agent has experienced the following schemas:

$$\begin{array}{ll} \langle b, travel, d\rangle & \langle d, turnAround, c\rangle \\ \langle c, turnRight, e\rangle & \langle e, travel, a\rangle \\ \langle a, turnAround, b\rangle & \end{array}$$

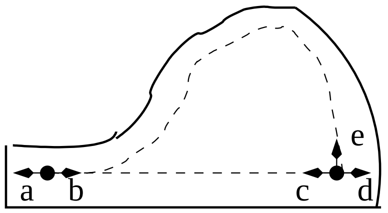


Figure 5.13: Distinctive state a is along two different paths. These two paths are declared convergent paths in the model of our theory.

From axiom 5.21 we know that exist paths Pa, $Pa1$ and directions dir, $dir1$ such that

$$along(b, Pa, dir) \wedge along(d, Pa, dir) \ ,$$

$$along(e, Pa1, dir1) \wedge along(a, Pa1, dir1) \ .$$

Moreover, from axiom 5.22 it follows that

$$along(b, Pa1, -dir1) \ .$$

We have two possible models for these schemas:

- **Model 1**. In this model $Pa \neq Pa1$. Consequently, $self\_intersecting = false$ and $convergent\_paths(Pa, Pa1)$ are the case.
- **Model 2**. In this model $Pa = Pa1$. Consequently, $self\_intersecting(Pa)$ and $convergent\_paths = false$ are the case.

We prefer model 1 over model 2 according to the circumscription policy 5.41. {*end of example*}

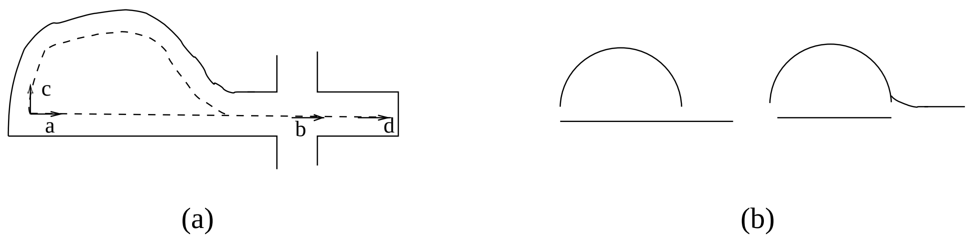


Figure 5.14: The environment in (a) can be described by two different topological maps. Either as a long straight path and a curved path, or as a short straight path and a long curved path.

Example 21

Whenever convergent paths exist, the theory will have multiple models. Consider the set of schemas (see figure 5.14) $\begin{array}{ll} \langle\, a, travel, b\,\rangle & \langle\, b, travel, d\,\rangle, \\ \langle\, a, turnLeft, c\,\rangle & \langle\, c, travel, b\,\rangle, \langle\, b, travel, d\,\rangle. \end{array}$ These schemas will have the following two models, which differ in the paths they use to explain the travel axioms:

1. a-b-d, c-b
2. c-b-d, a-b

Notice that in both cases the shorter path will not include the tuple b-d. This comes from the minimization of $along$. The minimization of $tpath$ also plays a role, since otherwise one could have a three paths model {a-b, c-b, b-d}.

Suppose one has the extra schema $\langle d, travel, e\rangle$. In this case the models will be extensions of the both above:

1. a-b-d-e, c-b
2. c-b-d-e, a-b

{*end of example*}

In order to compare any two possible maps, these maps must have a common sort of places and paths. Since a map can be arbitrarily large, no finite domain can be adequate and so we require the sorts of places and paths to be infinite. This requirement makes the the circumscription policy 5.41 to behave as expected. Consider the following example (see also theorem 5, page 46).

Example 22

Consider the schema $\langle a, travel, b\rangle$ where a and b have the same view. The intended model has one (topological) path and two (topological) places. One expects that the path is not circular (self-intersecting), and so the existence of two places. However, without requiring the existence of enough places, the following model is also possible:

places = {A}, tplace = {A}
paths = {Pa}, tpaths ={Pa}
$self\_intersecting(Pa)$
teq(a,b)
order(Pa,pos,A,A)
along(a,Pa,pos), along(b,Pa,pos)
at(a,A) at(b,A)

In this model, $self\_intersecting(Pa)$ has to be the case, since the universe of places only have one place. Notice that when comparing two models according to the circumscription policy 5.41, the universe of $paths$ and $places$ in the models has to be the same. One can vary the interpretation of $tpath$, $tplace$, and so on, but **not** the universe of $paths$ and $places$. The model above is ruled out by requiring the universe of $places$ to have enough places.
{*end of example*}

5.5.4 Explicit definitions

It is worth noticing that the blocks defining $convergent\_paths$ and $self\_intersecting$ can be transformed into first order explicit definitions of these predicates. In effect, in virtue of proposition 2 in [Lifschitz, 1994] we have that

$$\begin{aligned}&convergent\_paths(pa, pa1) \equiv \\ &\quad \exists ds, dir, dir1\,[along(ds, pa, dir) \wedge along(ds, pa1, dir1) \wedge \neg\,[pa = pa1 \wedge dir = dir1]]\ ,\end{aligned}$$

$$\begin{aligned}&self\_intersecting(pa) \equiv \\ &\quad \exists p, dir\,[order(pa, dir, p, p)] \\ &\quad \vee \\ &\quad \exists ds, ds', dir, dir1\,[\langle ds, Turn\_desc, ds'\rangle \wedge Turn\_desc \neq TurnAround \\ &\qquad\qquad \wedge\, along(ds, pa, dir) \wedge along(ds', pa, dir1)]\end{aligned}$$

$$\vee$$
$$\exists p, ds, ds', dir\, [at(ds,p) \wedge at(ds',p) \wedge \\ along(ds,pa,dir) \wedge along(ds',pa,dir) \wedge \neg teq(ds,ds')]$$

Instead of taking the above formulas as axioms, we prefer to define these predicates by giving "qualifications" of what they are, and let the minimization take the effect that given no additional information such qualification completely define the predicates. For instance, metrical information could be used to determine whether a path is self intersecting. This metrical qualification would be added to the block defining $self\_intersecting$. Had we decided to take the above formula as the definition of $self\_intersecting$ we would need to rewrite the corresponding definition.

Another advantage of using blocks to define predicates is when there is not an explicit formula defining the predicate. For instance consider the blocks defining $along$ and $order$ (page 41).

5.5.5 General SSH topological theory

Given the changes discussed in the previous sections, the AT_block for the SSH general topological theory is as follows:

$$AT\_block \;=$$
$$\{\; max\ teq :$$
$$teq\ is\ an\ equivalence\ relation\ ,$$
$$teq(ds_1,ds_2) \rightarrow View(ds_1,v) \equiv View(ds_2,v),$$
$$teq(ds_1,ds_2) \wedge \langle\, ds_1,a,ds_1' \,\rangle \wedge \langle\, ds_2,a,ds_2' \,\rangle \rightarrow teq(ds_1',ds_2'),$$
$$teq(ds_1,ds_2) \rightarrow \forall p\, [at(ds_1,p) \equiv at(ds_2,p)] \wedge \\ \forall pa, dir\, [along(ds_1,pa,dir) \equiv along(ds_2,pa,dir)]\ ,$$
$$\langle ds, turn, ds' \rangle \rightarrow \neg teq(ds,ds')\ ,$$
$$\langle ds, turnAround, ds' \rangle \wedge \langle ds, TurnAround, ds'' \rangle \rightarrow teq(ds',ds'')\ ,$$
$$\langle\, ds_1, turnAround, ds_2 \,\rangle \wedge \langle\, ds_2, turnAround, ds_3 \,\rangle \rightarrow teq(ds_1,ds_3)\ ,$$

$$at(ds,p) \rightarrow tplace(p),$$
$$\exists! p\, at(ds,p),$$
$$turn\_eq(ds_1,ds_2) \equiv \forall p\, [at(ds_1,p) \equiv at(ds_2,p)]\, ,$$

$$
\begin{aligned}
&\{min\ \ turn\_eq:\\
&\ \ teq(ds_1, ds_2) \wedge teq(ds_3, ds_4) \wedge \widehat{turn}(ds_2, ds_3) \rightarrow turn\_eq(ds_1, ds_4),\\
&\ \ turn\_eq(ds_1, ds_2) \wedge turn\_eq(ds_2, ds_3) \rightarrow turn\_eq(ds_1, ds_3)\\
&\}
\end{aligned}
$$

$$
\begin{aligned}
&along(ds, pa, dir) \rightarrow tpath(pa),\\
&\{\ min\ along:\\
&\ \ \langle\ ds, travel, ds'\ \rangle \rightarrow \exists pa,\ dir\ \left[along(ds, pa, dir) \wedge along(ds', pa, dir)\right],\\
&\ \ \langle ds, turnAround, ds'\rangle \rightarrow along(ds, pa, dir) \equiv along(ds', pa, -dir),\\
&\ \ teq(ds_1, ds_2) \rightarrow along(ds_1, pa, dir) \equiv along(ds_2, pa, dir)\\
&\}
\end{aligned}
$$

$$
\begin{aligned}
&\{\ min\ convergent\_paths:\\
&\ \ [along(ds, pa, dir) \wedge along(ds, pa1, dir1)\\
&\qquad \wedge\neg\,[pa = pa1 \wedge dir = dir1]] \rightarrow convergent\_paths(pa, pa1)\\
&\}\\
&\{min\ self\_intersecting\ :\\
&\ \ order(pa, dir, p, p) \rightarrow self\_intersecting(pa)\ ,\\
&\ \ [\langle ds, Turn\_desc, ds'\rangle \wedge Turn\_desc \neq TurnAround \wedge along(ds, pa, dir)\\
&\qquad\quad \wedge\ along(ds', pa, dir1)] \rightarrow self\_intersecting(pa)\ ,\\
&\ \ [at(ds_1, p) \wedge at(ds_2, p) \wedge along(ds_1, pa, dir) \wedge along(ds_2, pa, dir)\\
&\qquad\quad \wedge\ \neg teq(ds_1, ds_2)] \rightarrow self\_intersecting(pa)\\
&\}
\end{aligned}
$$

$$
\begin{aligned}
&\{\ min\ order:\\
&\ \ [\langle\ ds, travel, ds'\ \rangle \wedge at(ds, p) \wedge at(ds', q) \wedge\\
&\qquad along(ds, pa, dir) \wedge along(ds', pa, dir)] \rightarrow order(pa, dir, p, q),\\
&\ \ order(pa, pos, p, q) \equiv order(pa, neg, q, p),\\
&\ \ order(pa, dir, p, q) \wedge order(pa, dir, q, r) \rightarrow order(pa, dir, p, r)\\
&\}
\end{aligned}
$$

$$\{min\ on:\ at(ds,p) \wedge along(ds,pa,dir) \rightarrow on(pa,p)\ \}$$

$$\begin{aligned}
&on(pa,p) \wedge on(pa,q) \wedge tpath(pa) \rightarrow \\
&\quad \exists ds_1, dir_1, ds_2, dir_2\ [at(ds_1,p) \wedge along(ds_1,pa,dir_1) \wedge at(ds_2,q) \wedge \\
&\qquad\qquad along(ds_2,pa,dir_2) \wedge travel\_eq(ds_1,ds_2)]\,, \\
&\{min\ \ travel\_eq: \\
&\ \ \widehat{travel}(ds_1,ds_2) \rightarrow travel\_eq(ds_1,ds_2), \\
&\ \ \langle\, ds_1, turnAround, ds_2\,\rangle \rightarrow travel\_eq(ds_1,ds_2) \wedge travel\_eq(ds_2,ds_1) \\
&\ \ teq(ds_1,ds_2) \wedge teq(ds_3,ds_4) \wedge travel\_eq(ds_2,ds_3) \rightarrow travel\_eq(ds_1,ds_4), \\
&\ \ travel\_eq(ds_1,ds_2) \wedge travel\_eq(ds_2,ds_3) \rightarrow travel\_eq(ds_1,ds_3) \\
&\} \\
&\mathbf{circ}\ \ self\_intersecting \succ convergent\_paths \succ tpath \succ tplace\ \ \mathbf{var}\ \ \vec{SSHpred} \\
\}
\end{aligned}$$

5.6 Calculating the models of TT(E)

It is possible to calculate the models of $TT(E)$ by a logic program similar to the one used for $CT(E)$ (Section 4.4, Page 32). However, the number of grounding rules associated with such a program turns out to be prohibited for practical applications.

Fortunately, an algorithm for calculating the models of $TT(E)$ can be stated as a *"best first"* search. The states of the search correspond to partial models of $TT(E)$.[26] At each step of the search a schema $\langle\, ds, a, ds'\,\rangle$ has to be explained. Either the identity of ds' can be proved or a search branch is created for every previously known distinctive state ds'_i that cannot be proven to be different from ds'.[27] In the branch where $ds'_i \overset{teq}{=} ds'$ is the case, $ds'_j \overset{teq}{\neq} ds'$, $i \neq j$ are also asserted.[28] An additional brach is created where $ds' \overset{teq}{\neq} ds'_j$ are asserted. This branch represents the possibility that ds' is indeed different from previously known dstates. The next state to explore is the one that is minimum according to the order associated with the circumscription policy for $TT(E)$. This algorithm is described in fig-

[26]A partial model of $TT(E)$ is a model of $TT(E')$, for some $E' \subseteq E$.

[27]We assume that at each state of the search, the identity of the schema's context (i.e. ds in $\langle\, ds, a, ds'\,\rangle$) is known.

[28]The notation $ds_1 \overset{teq}{=} ds_2$ states that ds_1 and ds_2 are "equal" according to the equivalence relation teq. Recall that teq plays the role of equality in the theory $TT(E)$.

```
Find-Models (Schemas S)
{
  ;; S = $s_0, \ldots, s_n$ ; sequence of schemas such that
  ;;     $result(s_i) = context(s_{i+1})$
  ;;
  ;; pmodels-to-explore = ordered queue of partial models to explore.
  ;; models = list of total models for S.
  pmodels-to-explore = $\emptyset$ ;
  models = $\emptyset$ ;
  pmodel = create-new-pmodel(S);
  insert(pmodel,pmodels-to-explore) ;
  while pmodels-to-explore $\neq \emptyset$ do
    begin
      pmodel = get-next-pmodel(pmodels-to-explore);
      s = get-next-schema(pmodel);
      Explain(pmodel,s) ;
      if (inconsistent(pmodel) $\vee$ has-extensions(pmodel)) then skip;
      else if total-model(pmodel) then insert(pmodel, models);
      else insert(pmodel,pmodels-to-explore);
    end
  return models;
}
```

Figure 5.15: **Best first search algorithm used to calculate the models of TT(E)**. The ordered queue *pmodels-to-explore* contains *consistent* partial models (pmodels) to be expanded. At each step of the search, a minimal partial model is picked and the next schema from its list of associated schemas is explained. A pmodel has extensions when a branch has been created while explaining a schema. A pmodel is a *total-model* when it has no more schemas to explain. Figure 9.18 defines how a pmodel explains a schema and when extensions are created.

```
Explain (pmodel, s)
{ ;; s is a schema $\langle\, ds, a, ds' \,\rangle$
  candidates = {};
  branches = false;
  if $\neg$ known-result(pmodel,s)
  then begin
      candidates = possible-equal-dstates(pmodel,s);
      if candidates $\neq$ {}
        then branches = create-possible-extensions(pmodel,s,candidates)
    end
  if $\neg$ branches then Assert-schema (pmodel,s);
}
Known-result(pmodel, s)
{ ;; s is a schema $\langle\, ds, a, ds' \,\rangle$
  ;; The notation $obj \in pmodel$ indicates that object $obj$ is
  ;; known in the partial model $pmodel$.
  return $ds' \in pmodel \;\vee \exists\; ds^* \in pmodel \;\left[\langle\, ds^*, a, ds'^* \,\rangle \in pmodel \wedge ds^* \overset{teq}{=} ds\right]$ ;
}
Assert-schema (pmodel, s)
{ ;; s is a schema $\langle\, ds, a, ds' \,\rangle$. $ds$ is known in $pmodel$
  assert $s \in pmodel$;
  assert $ds' \in pmodel$;
  if $\neg$ known-result(pmodel,s)
   then       Create places and paths needed to explain $s$.
   else begin
     pick $ds'^*$ s.t. $\exists ds^* \in pmodel \;\left[ds^* \overset{teq}{=} ds \wedge \langle\, ds^*, a, ds'^* \,\rangle \in pmodel\right]$ ;
     assert $ds' \overset{pmodel}{=} ds'^*$ ;
   end
}
```

Figure 5.16: **Explaining a schema**. *known-result(pmodel,s* $= \langle\, ds, a, ds' \,\rangle$*)* is the case when the equality class for ds' can be deduced in the partial model $pmodel$. *Possible-equal-dstates(cntx,s)* returns dstates known in $pmodel$, having the same view as ds' and that cannot be proven different from ds' in $pmodel$. For each $ds'' \in candidates$, *create-possible-extensions(pmodel,s,candidates)* creates an extension of $pmodel$ where $ds' \overset{teq}{=} ds''$ is the case. If some of these extensions are consistent, then *create-possible-extensions* also creates an extension where ds' is different from the dstates in *candidates*. In this last case the function returns $false$ otherwise it returns $true$. If not extension is created, then s is asserted in $pmodel$. This accounts to declare ds' to be known in $pmodel$ and create the places and paths that explain s according to the axioms of the topological theory $TT(E)$.

ures 5.15 and 5.16. [29] See section 9.4.2, page 152, for a discussion on the implementation of this algorithm.

[29]A search state is implemented by a partial model, *pmodel*. Branches in the search are represented by creating *extensions* for the current search state (pmodel). That $pmodel'$ is an extension of $pmodel$ implies that $pmodel'$ inherits from $pmodel$ all known objects and facts. Partial models are described in page 155.

5.7 Summary

Actions performed by the agent are categorized at the SSH topological level into two classes: turn and travel. Turn actions leave the agent at the same place. Travel actions move the agent to a new place along a path. The SSH topological map is derived from the agent's experiences by creating the minimum set of paths and places explaining such experiences. In this chapter we have made precise this minimality criterion by specifying (via circumscription) the preferred models associated with the SSH topological level.

The concepts of *path* and *place* can be used to distinguish environment states that are not distinguishable by actions and views alone. In addition, we have illustrated how the agent's topological map changes as the agent acquires more experiences in the environment. These changes are non-monotonic as the agent can conclude, for example, that two previously believed different places are indeed the same (or vice versa).

It is possible that more than one topological map is consistent with the agent's set of experiences. In order to disambiguate the topological map, the agent must collect new experiences or rely on other sources of information to disambiguate its map (for example, metrical information or regions). It is also possible that there is no topological map consistent with the agent's experiences: this happens for example if the agent travels in a loop, violating our assumption that paths are not self-intersecting. This is not to say that we assume "rectilinear" environments where paths intersect at most in one place. As example 17 (page 55) shows, paths can intersect in more that one place. Another reason for not having a topological map are patterns of experience in which known different paths merge into the same distinctive state (i.e. convergent paths).

In section 5.5 we defined a general topological theory where self-intersecting and convergent paths were included. The general theory extended the more restricted one in the sense that a map according to the later is a map according to the former. The new theory captures the intuition that by default paths are not self-intersercting or convergent.

Finally, in section 5.6 we stated the problem of finding the models of $TT(E)$ as a "*best first*" search. In this search, the next state to explore is the one that is minimum according to the order associated with the circumscription policy for $TT(E)$. We defined the main elements of this search and postponed further details to the implementation chapter (section 9.4.2, page 152). The algorithm here described allows the agent to keep track of different possible models (maps) as well as to recover from distinctive state equality assumptions that make a model inconsistent.

Chapter 6

Boundary Regions

Topological paths play the role of *streets* in a city layout map. Streets are often used as a reference for specifying the location of a given place: a place will be either on the given street or in one of the “two sides” –left or right– of the street.

Mathematically, the concept of left and right of a SSH topological path is related to the topological one of the interior and exterior of a curve. While not all curves have a well defined interior and exterior (for example, consider a spiral, or a fractal curve), closed not self-intersecting[1] curves –*Jordan curves*– do have associated interior and exterior sets [Beardon, 1979].[2] Moreover, in order to go from the interior to the exterior (or vice versa) of the curve γ, one has to cross γ.[3] Our analogy of SSH topological paths and mathematical curves breaks down because in general the agent might be able to travel from one side of the path to the other without crossing the path. This can happen because of the agent’s inability to detect that it has crossed the path, or (more often) because paths are not long enough to divide the environment into two regions (for example, consider a dead-end street).

In order to determine boundary relations -the location of a place with respect to a path- we formally state the following heuristic. Suppose the agent is at an intersection on a given path, and it then turns right. If the agent now travels, any place it finds while traveling with no turns will be on the right of the starting path. While this heuristic draws the correct conclusion in a rectilinear environment, it may draw incorrect conclusions when paths are not straight. Consequently, we state our heuristic as a ‘defeasible” rule (default) so as not

[1]Except of course for the end points of the curve.

[2]When the curve is removed, the plane is divided into two disjoint connected sets.

[3]Given a curve γ' - a continuous function $\gamma' : [0,1] \rightarrow R^2$ - such that there exist t_1 and t_2, $\gamma'(t_1) \in$ interior of γ, $\gamma'(t_2) \in$ exterior of γ, then there exist t and t' such that $\gamma(t) = \gamma'(t')$.

to conclude a boundary relation when inconsistent sources of information exist. The next example illustrates this case.

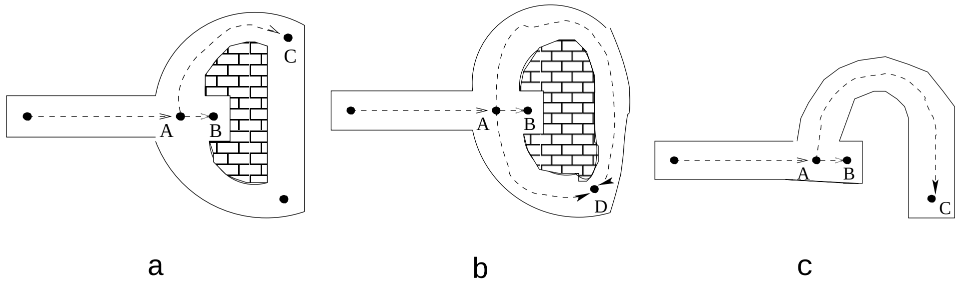


Figure 6.1: Different environments illustrating how our default to determine boundary relations work. In (a) we conclude by default that place C is to the left of the path from A to B. In (b) we conclude nothing about the location of place D with respect to this path. In (c) we conclude that place C is to the left of the path from A to B. This is the case since there is no information to conclude otherwise.

In the environment of Figure 6.1a we will conclude by default that place *C* is to the left of the path from *A* to *B*. We draw this conclusion since when the agent is at *A* facing in the direction of *B*, it turns left and travels with no turns up to place *C*.[4] The same argument in the case of the environment of Figure 6.1b will allow us to conclude by default that place *D* is to the left of the path from *A* to *B*. In this case, we assume that the environment "slowly" curves allowing the agent to keep traveling with no turns up to place *D*. However, if later the agent turns right at *A* and travels with no turns to *D*, by default we will conclude that place *D* is to the right of the path from *A* to *B*. These two default conclusions contradict each other and we solve this conflict by concluding nothing about the location of place *D* with respect to the path from *A* to *B*. Finally, in the environment of Figure 6.1c, the agent will conclude that place *C* is to the left of the path from *A* to *B*. This is the case since there is no information to conclude otherwise.

In order to determine boundary relations -the location of a place with respect to a path-, *turnRight* and *turnLeft* actions are used to define the relative orientation between paths at a given place (Section 6.1), relations that are then used to infer whether a place is on the left or the right of a given path (Section 6.2). The boundary relations inferred by an agent may not be complete: the agent does not necessarily know the location of each place with respect to each path. Nevertheless, the boundary relations inferred by the agent are useful to distinguish places otherwise not distinguishable by the SSH topological map as described so far (see Example 23, page 84).

[4]Travel with no turns indicates that the agent has kept traveling in a single path.

6.1 Qualitative orientation of paths at a place

We extend the topological level in order to represent the relative orientation among paths that intersect at a given place. We use the predicates

$$totheLeftOf(p, pa, dir, pa1, dir1) \quad , \quad totheRightOf(p, pa, dir, pa1, dir1)$$

to represent the facts that (i) *p* is a *place* on both paths, *pa* and *pa1*, and (ii), when the agent is at *place p* facing on the direction *dir* of *pa*, after executing a turn left (right) action, the agent will be facing on the direction *dir1* of *pa1* (see Figure 6.2, Page 80).

The predicates $totheLeftOf$ and $totheRightOf$ are derived from the actions performed by the agent at a place:[5]

$$\begin{aligned}
&\{min\ totheRightOf,\ min\ totheLeftOf\ : \\
&\quad [\langle ds, turnRight, ds1\rangle \wedge at(ds,p) \wedge along(ds,pa,dir) \wedge along(ds1,pa1,dir1)] \\
&\qquad\qquad \rightarrow totheRightOf(p,pa,dir,pa1,dir1), \\
&\quad [\langle ds, turnLeft, ds1\rangle \wedge at(ds,p) \wedge along(ds,pa,dir) \wedge along(ds1,pa1,dir1)] \\
&\qquad\qquad \rightarrow totheLeftOf(p,pa,dir,pa1,dir1). \\
&\}
\end{aligned} \tag{6.1}$$

6.2 Left and Right of a path

A path has associated two regions: the places to the left of the path and the places to the right of the path.[6] We use the predicates

$$leftOf(pa, dir, lr) \quad , \quad rightOf(pa, dir, rr)$$

to denote that *region* lr (rr) is the left (right) region of path *pa* with respect to the path's direction *dir*. The properties of these predicate are as follows:

$$\exists! lr\,\{leftOf(pa,dir,lr)\}\,,\quad \exists! rr\,\{rightOf(pa,dir,rr)\} \tag{6.2}$$

$$leftOf(pa,dir,r) \equiv rightOf(pa,-dir,r) \tag{6.3}$$

$$\begin{aligned}
&\{\,min\ is\_region\ : \\
&\qquad LeftOf(pa,dir,lr) \rightarrow is\_region(lr) \\
&\}
\end{aligned} \tag{6.4}$$

$$leftOf(pa,dir,lr) \wedge leftOf(pa1,dir1,lr) \rightarrow pa = pa1 \tag{6.5}$$

[5]It is possible that the agent learns these relations by being told. In the presence of no other information, the implications in block 6.1 become equivalences (proposition 2 in [Lifschitz, 1994]).

[6]$Regions$ are set of places or regions. This way, regions form a containment hierarchy that allows the agent to reason at different levels of detail about its spatial representation. In Chapter 8 we present a complete formalization of regions in the context of the SSH.

Axiom 6.2 states the existence and uniqueness of a path's left/right regions. The domain of $leftOf$ is restricted by block 6.4 and axiom 6.5. Since left/right regions of a path interchange when changing the path direction (axiom 6.3), constraining the domain of $leftOf$ imposes similar constraints on the domain of $rightOf$.

We use the predicate **in_region(p,r)** to indicate that *place* p is in *region* r. The domain of *in_region* is constrained by axiom 6.6. The properties of in_region are defined in block 6.7. Using this predicate, we state that a path has associated three disjoint set of places: the places on the path, and the places to the left/right of the path(Axioms 6.9 and 6.10).[7] Boundary relations are derived according to axiom 6.11 and 6.12 (see Figure 6.2):

$$in\_region(p,r) \rightarrow is\_region(r)\,, \tag{6.6}$$

$$\{\ min\ in\_region\ : \tag{6.7}$$

$$\{\ in\_region : \tag{6.8}$$

$$on(pa,p) \wedge leftOf(pa,dir,lr) \rightarrow \neg in\_region(p,lr), \tag{6.9}$$

$$[leftOf(pa,dir,lr) \wedge rightOf(pa,dir,rr) \wedge in\_region(p,lr)] \rightarrow \neg in\_region(p,rr)\,, \tag{6.10}$$

$$\begin{aligned}&[totheRightOf(p1,pa,dir,pa1,dir1) \wedge order(pa1,dir1,p1,p)\wedge \\ &\qquad rightOf(pa,dir,rr) \wedge \neg \mathbf{Ab(pa,p)}] \rightarrow in\_region(p,rr),\end{aligned} \tag{6.11}$$

$$\begin{aligned}&[totheLeftOf(p1,pa,dir,dir1,pa1) \wedge order(pa1,dir1,p1,p)\wedge \\ &\qquad leftOf(pa,dir,lr) \wedge \neg \mathbf{Ab(pa,p)}] \rightarrow in\_region(p,lr)\end{aligned} \tag{6.12}$$

$$\}$$

$$\}$$

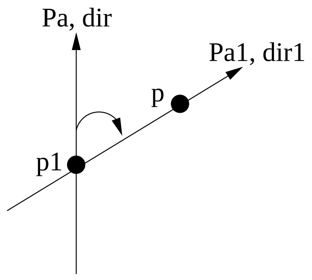


Figure 6.2: Path *Pa1* is to the right of path *Pa* at place *p1*. Place *p* is after place *p1* on path *pa1*. By default, we conclude that place *p* is to the right of path *pa*.

[7]The symmetry between $leftOf$ and $rightOf$ defined by axiom 6.3 let us write our axioms in terms of only one of these predicates. For example, notice that axioms 6.3 and 6.9 imply that

$$on(pa,p) \wedge rightOf(pa,dir,rr) \rightarrow \neg in\_region(p,rr)$$

The outer block 6.7 states that normally boundary relations are false. This is the case since by default the agent does not know the location of a place with respect to a given path. The inner block 6.8 states under what conditions the agent can derive a boundary relation. For instance, according to axiom 6.11, if at place *p1* path *pa1* is to the right of path *pa*, and place *p* is after *p1* on path *pa1*, then normally it is the case that *p* is on the right of *pa* (see Figure 6.2).[8] The predicate **Ab** inside block 6.8 is the standard "abnormality" predicate used to represent defaults in circumscriptive theories ([McCarthy, 1980, Lifschitz, 1994], see Appendix C, Page 219).

While block 6.7 defines the extent of the predicate in_region, it does not express our preference for models where in_region is maximal. We need to include in_region in our circumscription policy, as shown in the next section. In order to understand why this is the case, assume $Ab = false$ inside block 6.8. Consequently, block 6.7 becomes

$$
\begin{array}{l}
\{\; min\; in\_region\; : \\
\quad on(pa,p) \wedge leftOf(pa,dir,lr) \rightarrow \neg in\_region(p,lr)\;, \\
\quad [leftOf(pa,dir,lr) \wedge rightOf(pa,dir,rr) \wedge in\_region(p,lr)] \rightarrow \neg in\_region(p,rr)\;, \\
\quad [totheRightOf(p1,pa,dir,pa1,dir1) \wedge order(pa1,dir1,p1,p) \wedge \\
\qquad\qquad\qquad rightOf(pa,dir,rr)] \rightarrow in\_region(p,rr)\;, \\
\quad [toTheLeftof(p1,pa,dir,dir1,pa1) \wedge order(pa1,dir1,p1,p) \wedge \\
\qquad\qquad\qquad leftOf(pa,dir,lr)] \rightarrow in\_region(p,lr) \\
\}
\end{array}
$$

This block can be further simplified as follows (see Propositions 2 and 4 in [Lifschitz, 1994]):

$$
\begin{array}{l}
on(pa,p) \wedge leftOf(pa,dir,lr) \rightarrow \neg in\_region(p,lr)\;, \\
[leftOf(pa,dir,lr) \wedge rightOf(pa,dir,rr) \wedge in\_region(p,lr)] \rightarrow \neg in\_region(p,rr)\;, \\
in\_region(p,r) \\
\equiv \\
\exists p1,pa,dir,pa1,dir1\{ \\
\quad [totheRightOf(p1,pa,dir,pa1,dir1) \wedge order(pa1,dir1,p1,p) \wedge rightOf(pa,dir,r)] \\
\qquad \vee \\
\quad [toTheLeftof(p1,pa,dir,dir1,pa1) \wedge order(pa1,dir1,p1,p) \wedge leftOf(pa,dir,r)] \\
\qquad\qquad\qquad \}\;.
\end{array}
$$

Since block 6.7 "normally" is equivalent to the statements above, and so, it "normally"

[8]See appendix C, page 219, for a similar formalization of the standard example: objects normally do not fly; birds normally do.

reduces to an explicit definition for in_region, block 6.7 does not express our preference for models where in_region is maximal, preference that has to be expressed elsewhere (see next section).[9]

6.2.1 Adding boundary relations to the topological map

In order to use boundary relations when defining the SSH topological map, axioms 6.1-6.12 are included inside the block **AT_block** (see Section 5.1, page 41), and the new circumscription policy becomes[10]

$$\textbf{circc} \;\; tpath \succ \neg \textbf{in\_region} \succ tplace \;\; \textbf{var} \;\; new\vec{SSH}pred$$

where $new\vec{SSH}pred$ stands for the tuple of predicates

$$\begin{aligned}&\langle \;\; at, along, order, on, teq, turn\_eq, travel\_eq,\\ &\quad \mathbf{totheRightOf}, \mathbf{totheLeftOf}, \mathbf{leftOf}, \mathbf{rightOf}, \mathbf{is\_region}\\ &\rangle \;.\end{aligned}$$

The circumscription policy states that axioms 6.11 and 6.12 should be used to draw conclusions even at the expense of having more places on the map. This is achieved by maximizing in_region over $tplace$ in the circumscription policy. Next we show the new theory.

$$\begin{aligned}
&AT\_block = \\
&\quad \{ \;\; max\; teq \;\; : \\
&\qquad \textbf{Axioms 6.1-6.12}, \\
&\qquad teq\; is\; an\; equivalence\; relation \;, \\
&\qquad teq(ds_1, ds_2) \rightarrow View(ds_1, v) \equiv View(ds_2, v), \\
&\qquad teq(ds_1, ds_2) \wedge \langle\, ds_1, a, ds_1' \,\rangle \wedge \langle\, ds_2, a, ds_2' \,\rangle \rightarrow teq(ds_1', ds_2'), \\
&\qquad teq(ds_1, ds_2) \rightarrow \forall p\, [at(ds_1, p) \equiv at(ds_2, p)] \wedge \\
&\qquad\qquad\qquad\qquad \forall pa, dir\, [along(ds_1, pa, dir) \equiv along(ds_2, pa, dir)]\,, \\
&\qquad \langle ds, turn, ds' \rangle \rightarrow \neg teq(ds, ds') \;, \\
&\qquad \langle ds, turnAround, ds' \rangle \wedge \langle ds, TurnAround, ds'' \rangle \rightarrow teq(ds', ds'') \;, \\
&\qquad \langle\, ds_1, turnAround, ds_2 \,\rangle \wedge \langle\, ds_2, turnAround, ds_3 \,\rangle \rightarrow teq(ds_1, ds_3) \;, \\
&\\
&\qquad at(ds, p) \rightarrow tplace(p), \\
&\qquad \exists! p\; at(ds, p),
\end{aligned}$$

[9]Notice that the statements above are the ones characterizing boundary relations in “rectilinear” environments.

[10]See circumscription notation in page 222.

$turn\_eq(ds_1, ds_2) \equiv \forall p\, [at(ds_1, p) \equiv at(ds_2, p)]\,,$

$\{min\ \ turn\_eq:$

$\quad teq(ds_1, ds_2) \wedge teq(ds_3, ds_4) \wedge \widehat{turn}(ds_2, ds_3) \rightarrow turn\_eq(ds_1, ds_4),$

$\quad turn\_eq(ds_1, ds_2) \wedge turn\_eq(ds_2, ds_3) \rightarrow turn\_eq(ds_1, ds3)$

$\}$

$along(ds, pa, dir) \rightarrow tpath(pa),$

$\{\ min\ along:$

$\quad \langle\, ds, travel, ds'\,\rangle \rightarrow \exists pa,\ dir\ [along(ds, pa, dir) \wedge along(ds', pa, dir)]\,,$

$\quad \langle ds, turnAround, ds'\rangle \rightarrow along(ds, pa, dir) \equiv along(ds', pa, -dir),$

$\quad teq(ds_1, ds_2) \rightarrow along(ds_1, pa, dir) \equiv along(ds_2, pa, dir)$

$\}$

$along(ds, pa, dir) \wedge along(ds, pa1, dir1) \rightarrow pa = pa1 \wedge dir = dir1,$

$at(ds_1, p) \wedge at(ds_2, p) \wedge along(ds_1, pa, dir) \wedge along(ds_2, pa, dir) \rightarrow teq(ds_1, ds_2),$

$[\langle ds, turn\_desc, ds'\rangle\ \wedge\ turn\_desc \neq turnAround \wedge$

$\qquad along(ds, pa, dir) \wedge along(ds', pa1, dir1)] \rightarrow pa \neq pa1,$

$\{\ min\ order:$

$\quad [\langle\, ds, travel, ds'\,\rangle \wedge at(ds, p) \wedge at(ds', q) \wedge$

$\qquad along(ds, pa, dir) \wedge along(ds', pa, dir)] \rightarrow order(pa, dir, p, q),$

$\quad order(pa, pos, p, q) \equiv order(pa, neg, q, p),$

$\quad order(pa, dir, p, q) \wedge order(pa, dir, q, r) \rightarrow order(pa, dir, p, r)$

$\}$

$\neg order(pa, dir, p, p),$

$\{min\ on:\ \ at(ds, p) \wedge along(ds, pa, dir) \rightarrow on(pa, p)\ \}$

$on(pa, p) \wedge on(pa, q) \wedge tpath(pa) \rightarrow$

$\qquad \exists ds_1, dir_1, ds_2, dir_2\ [at(ds_1, p) \wedge along(ds_1, pa, dir_1) \wedge at(ds_2, q) \wedge$

$\qquad\qquad\qquad along(ds_2, pa, dir_2) \wedge travel\_eq(ds_1, ds_2)]\,,$

$\{min\ \ travel\_eq:$

$\quad \widehat{travel}(ds_1, ds_2) \rightarrow travel\_eq(ds_1, ds_2),$

$\quad \langle\, ds_1, turnAround, ds_2\,\rangle \rightarrow travel\_eq(ds_1, ds_2) \wedge travel\_eq(ds_2, ds_1),$

$\quad teq(ds_1, ds_2) \wedge teq(ds_3, ds_4) \wedge travel\_eq(ds_2, ds_3) \rightarrow travel\_eq(ds_1, ds_4),$

$travel\_eq(ds_1, ds_2) \wedge travel\_eq(ds_2, ds_3) \rightarrow travel\_eq(ds_1, ds_3)$
}

circc $tpath \succ \neg$**in_region** $\succ tplace$ **var** $newS\vec{S}Hpred$
}

By maximizing the extent of in_region at the expense of having possibly more places, boundary relations determine distinctions among environment states that could not be derived from the connectivity of places alone. The next example illustrates the case.

Example 23

Consider an agent visiting the different corners of a square room in the order suggested by figure 6.3a. In addition, suppose the agent's sensory apparatus allows it to define *views* by characterizing the direction of walls and open space. Accordingly, the agent experiences *four* different views, *v1-v4*, in this environment (see Figure 6.4).

The agent's experiences, *E*, in this environment are:

$View(ds1, v1)$	$View(ds2, v2)$	$View(ds3, v1)$
$View(ds4, v2)$	$View(ds5, v1)$	
$\langle ds1, turnRight, ds2\rangle$	$\langle ds2, travel, ds3\rangle$	$\langle ds3, turnRight, ds4\rangle$
$\langle ds4, travel, ds5\rangle$		

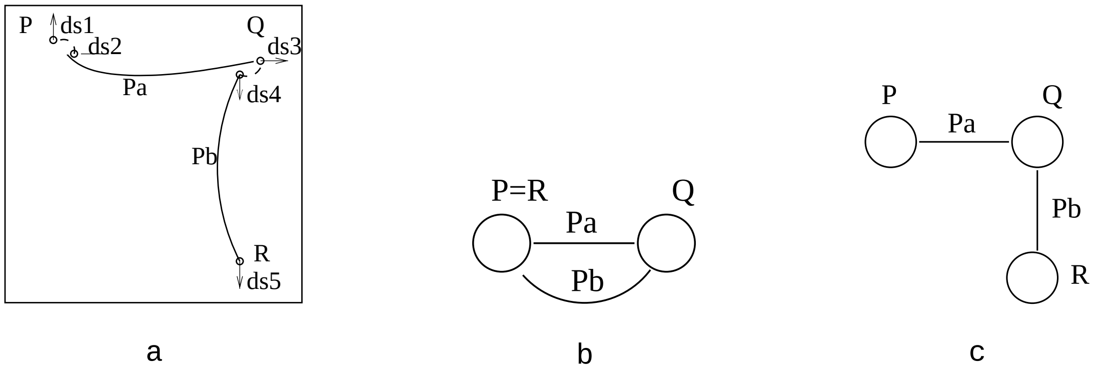


Figure 6.3: (a) The figure shows the sequence of actions followed by an agent while navigating a square room. Starting at distinctive state ds1, distinctive states are visited in the order suggested by their number. Dashed lines indicate Turn actions. Solid lines indicate Travel actions. (b) and (c) depict the resulting topological map without and using boundary regions, respectively.

View	Left	Ahead	Right	Behind
v1	wall	wall	open	open
v2	wall	open	open	wall
v3	open	wall	wall	open
v4	open	open	wall	wall

Figure 6.4: Definition of views v1-v4. Each view is characterized by the direction of walls and open space.

Suppose that the agent does not use boundary regions when building the topological map. Then the minimal topological model associated with *E* has two paths[11] and two places. In this model, $teq(ds1, ds5)$ is the case. The environment looks perfectly symmetric to the agent (Figure 6.3b).!!

Suppose now that the agent relies on boundary regions. Let *P*, *Q*, *R*, be the topological places associated with *d1*, *d3* and *d5* respectively. From Axiom 5.21, let *Pa*, *Pb*, dir_a and dir_b be such that

$$order(Pa, dir_a, P, Q)\ ,\ along(ds2, Pa, dir_a),\ along(ds3, Pa, dir_a),$$

$$order(Pb, dir_b, Q, R),\ along(ds4, Pb, dir_b),\ along(ds5, Pb, dir_b)\ ,$$

are the case. From block 6.1 we conclude that

$$totheRightOf(Q, Pa, dir_a, Pb, dir_b)\ \ .$$

In the proposed model, the extent of in_region is maximized by declaring

$$Ab = false$$

inside block 6.8 and consequently from axiom 6.11 we conclude

$$in\_region(R, right(Pa, dir_a))$$

where $right(Pa, dir_a)$ denotes the right region of Pa when facing dir_a.[12] Moreover, from block 6.7 we deduce

$$in\_region(p, r) \equiv [p = R \wedge r = right(Pa, dir_a)]\ \ .$$

Finally, from axiom 6.9 we conclude

$$P \neq R$$

[11]Notice that from $\langle ds3, turnRight, ds4\rangle$ and axiom 5.26 we can deduce that $Pa \neq Pb$ in figure 6.3b.
[12]Axiom 6.2 guarantees that there exists a unique such region.

since $on(Pa, P)$ is the case. The resulting topological map is depicted in Figure 6.3c.
{*end of example*}

Boundary relations are in general not enough to distinguish different environment states. The example below illustrates the case.

Example 24

Consider the previous example, and suppose the agent continues the navigation by visiting the distinctive states $ds6$ and $ds7$ (see Figure 6.5).

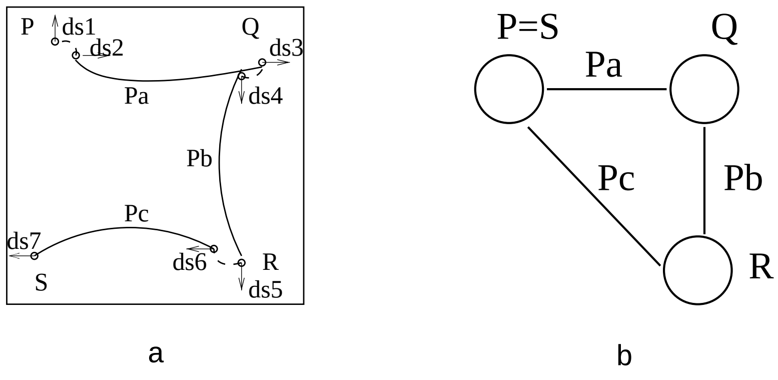


Figure 6.5: The agent continues the exploration of the square room by traveling to *ds7*. Although *ds7* and *ds1* occur at different environment states, the agent concludes that $teq(ds1, ds7)$ and associates with this environment the topological map depicted on (b).

The new set of experiences *E* will be extended by including the following formulae:

$$View(ds6, v2) \quad View(ds7, v1) \quad \langle ds5, turnRight, ds6\rangle \quad \langle ds6, travel, ds7\rangle \quad .$$

Let S denote the topological place associated with $ds7$. A minimal topological model associated with *E* has $P = S$ and $teq(ds7, ds1)$ (see Figure 6.5b). Notice that once the agent turns right at $ds5$ and travels to $ds7$, it cannot deduce that it is still to the right of the path connecting places P and Q.
{*end of example*}

The problem above arises because the agent's sensory capabilities are so impoverished that many distinctive states are indistinguishable (i.e. there is no view-action-view sequence that distinguishes them). Even a slightly richer sensory system (e.g. including a compass) would eliminate the problem in example 24, and make sensory aliasing much less common globally. The problem cannot be eliminated entirely through better sensors, since some environment states may be genuinely indistinguishable by any given set of sensors.

How can the agent realize that *ds7* and *ds1* in the previous example are indeed different environment states?. This apparently easy problem turns out to be a hard one. Next we summarize different approaches to solve this problem:

- The agent can gather more information as in the *Rehearsal procedure* proposed in [Kuipers and Byun, 1988]. In order to disambiguate two distinctive states the agent finds a sequence of actions and views that prove (or disprove) that the agent is at a given distinctive state. Once that sequence is found, the agent performs the actions in the sequence and checks whether it obtains the expected views. If some of the views obtained during the navigation do not match the expected views, the agent concludes that it was not at the hypothesized distinctive state. If all of the views match the expected views, the agent cannot conclude anything about its starting distinctive state. This method has three drawbacks: (i) the disambiguating sequence may not exist (as in Example 24), (ii) the method is partially conclusive, and (iii), it requires physical navigation. While a partially conclusive method, richer sensors will increase the effectiveness of the rehearsal procedure.

  The rehearsal procedure is closely related to the problem of generating counterexamples that can distinguish states in a deterministic finite state automata. If such sources of counterexamples exist and the agent has the ability to at will go to a given but fixed state, then learning the smallest automaton (w.r.t. the number of states) consistent with a given set of actions/output pairs can be done in polynomial time [Angluin, 1987]. Without particular assumptions about the environment or the agent's perceptual abilities, the problem of finding this smallest automaton is NP-complete ([Angluin, 1978, Gold, 1978]). The reader is referred to [Basye *et al.*, 1995] for an example of how to use automata to model dynamical systems.

- In order to distinguish distinctive states one could "mark" the current state and then attempt to disambiguate distinctive states as in the rehearsal procedure [Dudek *et al.*, 1991]. "Marking" the environment corresponds to associate a unique view with the current environment state. This could be achieved by physically changing the environment (as in leaving a "I was here" note) or finding a feature not observed in any other state. Here is how this method work in the case of Example 24.

Once at *ds7* the agent wants to prove the hypothesis $ds1 = ds7$. Assuming that this is the case, after leaving a marker in the current location, and executing the sequence of actions

$$turnRight\ ;\ travel\ ;\ turnRight\ ;\ travel\ ;\ turnRight\ ;\ travel\ ,$$

the agent should be back at $ds1$ and recognize the marker it left there (see Figure 6.5). In our example, after executing this sequence of actions the agent will be at *ds5* and will not find the marker. Consequently, the agent will deduce that $ds1 \neq ds7$. Now the problem is to go back to $ds7$ and remove the marker left there. In order to do so, one has to assume that actions are *reversible*, and the agent knows how to reverse a sequence of actions.

- Use metrical information (i.e. the amount of turns and travel) to integrate the different distinctive states (or at least a set of them) into a common frame of reference. By comparing the position of *ds1* and *ds7* in such a frame of reference, the agent can conclude whether they are the same environment state.

  The problem with this approach is the uncertainty (error) associated with metrical information. In the presence of uncertainty, metrical information allows the agent with high confidence to decide whether two distinctive states are *not* the same. The agent will require better metrical estimates to decide if two distinctive states are the same. In current approaches to do so, the agent will navigate various times around the environment in order to wash out error measurements and have better certainty about the position of distinctive states in a common frame of reference (see [Engelson and McDermott, 1992b, Koenig and Simmons, 1996, Shatkay and Kaelbling, 1997]). In Chapter 7 we illustrate the use of metrical information in the context of the SSH. In particular, if metrical and topological information are consistent with the hypothesis that the agent is back to a particular distinctive state, by default we then conclude that this is the case.

6.3 Summary

In this chapter we have introduced the left and right regions associated with a path, and show how the agent determines boundary relations –the location of a place with respect to a path–. Using boundary relations the agent can distinguish places not distinguishable from connectivity relations alone (see Example 23).[13] We use "defeasible" (default) rules to state

[13]In our previous work on boundary relations, we assumed that the map was given and show how the boundary relations were acquired [Remolina and Kuipers, 1998a]. Here, we have extended our formalization to use

how the agent acquires these boundary relations (Axioms 6.11-6.12). Being defaults, these rules allow the agent to deal with inconsistent information while not restricting their applicability to “rectilinear” environments (see Figure 6.1).

We have illustrated the case (Example 24) where poor sensory capabilities in addition to the symmetry of the environment will prevent the agent from differentiating some environment states. In these cases, the SSH topological map will not capture the correct topology of the environment. Nevertheless, the SSH topological map will have the minimum set of paths and places that explain the different view-action-view sequences experienced by the agent in the environment.

Finally, we have presented different alternative methods for disambiguating distinctive states: (i) the *rehearsal procedure*, where the agent tries to find a sequence of actions that will differentiate two distinctive states, (ii) using “markers” to indicate where the agent has been, and (iii) using metrical information. While theoretically none of these methods solves the problem of differentiating distinctive states, they could be very effective in helping the agent to discard some of the otherwise possible maps consistent with its experiences.

boundary relations during the construction of the SSH topological map.

Chapter 7

Using Metrical Information

Quantitative spatial information is represented at each level of the SSH, from local analog maps at the control level, to action magnitudes at the causal level, to local headings and distances at the topological level [Kuipers, 2000]. In order for an agent to reason about orientation and distance between places, places must be located in a common frame of reference.[1] In this chapter we discuss different frames of reference that can be associated with SSH objects (i.e. distinctive states, places, path, regions). In particular,

- Each path has associated a one dimensional frame of reference which assigns a position to each place in the path.
- Each place has associated a radial frame of reference which assigns a heading (angle) to each path the place belongs to.
- Each path has associated a frame of reference which assigns to certain places a location from $\{$*left*, *right*, *on*$\}$.[2]
- Regions or places might have associated two dimensional frames of reference which assign real valued tuples to certain places.

As *positions* and *headings* are derived from noisy data, there is uncertainty associated with their real values. Different representations for this uncertainty are possible: *intervals, probability distribution functions,* etc. As the agent repeatedly navigates among

[1]A location in a frame of reference can be described by a real value, an angle, a real valued tuple, a discrete value (e.g. *left*, *right*), etc.

[2]This corresponds to the boundary regions described in chapter 6. Notice that this is not a *metrical* frame of reference.

the same places and paths, new measure estimates are taken into account to update the uncertainty associated with positions and headings.[3] In our current work we use *intervals* to represent uncertainty in position and headings. However, the methods we define to integrate multiple frames of reference are general to other forms of representing uncertainty.

In addition to frames of reference, the SSH metrical level can accommodate more sophisticated representations of metrical information. For example, each place could have associated an *occupancy grid* [Elfes, 1987, Borestein and Koren, 1991, Thrun, 1998] where the position and extent of objects in the place's local neighborhood could be represented. The origin and scale of such an occupancy grid will coincide with those of the frame of reference associated with the place. In this way, paths headings could be related to the occupancy grid.

This chapter is organized as follows: in section 7.1 we define how to represent metrical information associated with schemas. The different kind of frames of reference used by the SSH are then presented in section 7.2. In section 7.3 we define how topological and metrical information are combined while the agent builds its map. In particular, we define what it means for a topological map to be consistent with its metrical information. Finally, in section 7.4 we list different techniques that are relevant when building the SSH metrical map. In particular, we adapt a basic constraint propagation algorithm in order to create two dimensional frames of reference preserving the estimated distance and relative orientation between consecutive places in a path.

7.1 Schemas, representing uncertainty

Action executions have associated metrical information representing the observed magnitude of the action. For instance, after traveling the agent may have an estimate of the distance between the "end places" of the travel action, and after turning, the agent may have an estimate of the angle turned.[4] Uncertainty associated with these estimates must be taken into account when using metrical information in order to derive relations among SSH objects. Hereafter we assume that uncertainty is represented by *positive real valued closed intervals*.

[3]For example, Kuipers and Byun [Kuipers and Byun, 1988, Kuipers and Byun, 1991] use a scalar Kalman-filter in order to update the distance between two places in a path.

[4]Notice that different kind of metrical estimates could be associated with a travel or turn action. For example, the agent could measure the arc length associated with a travel action. In addition, it could measure the minimum distance to an object on the left and the right sides at each point along the trajectory associated with a travel action [Kuipers, 2000].

We use the predicate

$$action\_execution(s, Int) \ ,$$

to state that the *interval* Int represents an estimate of the metrical information about the execution of the action associated with schema *s*. How the estimates are to be interpreted depends on the type of action (turn or travel) the schema refers to. In the next sections we will describe how to do so. We do not further restrict the domain of the predicate $action\_execution$. In particular, it is possible to have none or more than one estimate associated with the same schema.

We use the notation

$$\langle ds, (type\ Int), ds' \rangle \ ,$$

where *type* is *travel* or *turn*, as an abbreviation for the formula

$$\exists s, a\ \{CS(s, ds, a, ds') \wedge action\_type(a, type) \wedge action\_execution(s, Int)\} \ .$$

7.2 Frames of reference

In this section we consider one dimensional frames of reference which assign positions to places on a path, radial frames of reference which assign angles to the different paths that intersect at a given place, and two dimensional frames of reference which assign positions to arbitrary set of places. Although we talk about frames of reference, we do not have a sort for them. We represent them implicitly by associating them with places or paths as explained below.

7.2.1 One dimensional frames of reference

At the SSH metrical level each path has associated a one dimensional frame of reference which assigns a location to each place on the path. This location is a real number,[5] representing the "distance" with respect to an arbitrary but fixed place on the path. The positions of places in a path are derived from estimates acquired when traveling among places on the

[5]This real value represents a quantity whose magnitude is derived by the robot while navigating the environment. The units of this quantity can be *meters, feet,* or *number of wheel rotations*. Hereafter, we assume that all quantities are given in the same units.

path.[6] Next we formalize these ideas.

The position of a place on a path is represented by the predicate

$$position1(path, place, position) \ .$$

Positions along a path are unique and only assigned to places belonging to the path. Formally,

$$position1(pa, p, pos) \wedge position1(pa, p, pos') \rightarrow pos = pos' \ , \tag{7.1}$$

$$position1(pa, p, pos) \rightarrow on(pa, p) \ . \tag{7.2}$$

The distance between two places in a path is defined as the absolute value of the difference between their corresponding positions on the path. We use the predicate

$$path\_distance(pa, p, q, d)$$

to represent the fact that the distance between places *p* and *q* on path *pa* is *d*. The predicate $path\_distance$ is defined as follows:

$$\begin{aligned} & path\_distance(pa, p, q, d) \equiv \\ & \quad \exists pos_p,\ pos_q\ \{position1(pa, p, pos_p) \wedge position1(pa, q, pos_q) \wedge d = |pos_p - pos_q|\} \ . \end{aligned} \tag{7.3}$$

Estimates of the distance between places on a path are gathered while the agent navigates the environment. The predicate

$$path\_distance^{\approx}(pa, p, q, I_d)$$

is used to represent the fact that the closed interval I_d is an estimate of the distance between places *p* and *q* on path *pa*. Distance estimates are derived from experiences of the robot in the environment. Distance estimates are "compounded" to derive new estimates from known ones. Formally,

$$\begin{aligned} & \{\ min\ path\_distance^{\approx} : \\ & \quad [\langle\ ds, (travel\ I_d), ds'\ \rangle \wedge at(ds, p) \wedge at(ds', q) \wedge along(ds, pa, dir) \wedge \qquad (7.4) \\ & \qquad along(ds', pa, dir)] \rightarrow path\_distance^{\approx}(pa, p, q, I_d) \ , \\ & \quad [order(pa, dir, p, q) \wedge order(pa, dir, q, r) \wedge path\_distance^{\approx}(pa, p, q, I_{pq}) \wedge \qquad (7.5) \\ & \qquad path\_distance^{\approx}(pa, q, r, I_{qr})] \rightarrow path\_distance^{\approx}(pa, p, r, I_{pq} + I_{qr}) \\ & \} \end{aligned}$$

[6] The units of a 1-D frame of reference do not need to be externally meaningful. They must only be consistent over time for the individual agent. One dimensional positions are "internal quantities" (as distinct from nominal, ordinal, or ratio [Stevens, 1946]). That is, differences can be computed and added to positions, but multiplication and division are not meaningful since the origin is an arbitrary landmark, not a true "zero" quantity.

where the addition of intervals is defined in the usual way: $[a, b] + [c, d] = [a + c, b + d]$. Finally, distance estimates are "merged" in order to have the "best" estimate associated with a distance. We use the predicate $path\_distance^{\otimes}(pa, p, r, I_d)$ defined below in order to denote the merging of distance estimates:

$$path\_distance^{\otimes}(pa, p, r, I) \equiv_{def} I = \cap\{I_{est} \ : \ path\_distance^{\approx}(pa, p, q, I_{est})\} \ . \tag{7.6}$$

The distance between places on a path has to be *compatible* with all of its estimates. Formally,[7]

$$path\_distance^{\otimes}(pa, p, q, I_d) \rightarrow \exists d \in I_d \ path\_distance(pa, p, q, d) \ . \tag{7.7}$$

The operations of compounding and merging are standard operations in order to propagate uncertainty about the real value of a variable [Smith and Cheeseman, 1986]. They take different forms depending on how one represents uncertainty as well as on the dimensionality of the variables' domains. In our case, these operations take the form of adding and intersecting intervals, respectively.[8] Our description of how to assign positions to places on a path is a particular instance of a more general problem where given a set of related variables as well as uncertainty about the values of the variables, one has to determine the "best" estimate possible for the real value of the variables, while preserving the relations among them [Smith and Cheeseman, 1986, Durrant-Whyte, 1988b, Moutarlier and Chatila, 1989].

7.2.2 Radial frames of reference

Each place has a local frame of reference w.r.t. which path headings are associated. This information is represented by the predicate

$$radial(p, pa, dir, h)$$

whose intended meaning is *when the agent is located at place* p*, path* pa *could be followed in direction* dir *by facing the heading* h *w.r.t. the radial frame of reference local to* p. Headings take values in $[0, 2\pi)$.

[7]Axiom 7.7 also requires that the different distance estimates should be *consistent*, $path\_distance^{\otimes}(pa, p, q, I_d) \rightarrow I_d \neq \emptyset$.

[8]If $x_1, \ldots, x_n$ are n independent normally distributed measurements of a single quantity x, with known uncertainties $\sigma_1, \ldots, \sigma_n$, then the best estimate for the true value of x is the *weighted average*

$$x_{avg} = \frac{\sum w_i x_i}{\sum w_i}$$

where $w_i = \frac{1}{\sigma_i^2}$ [Taylor, 1997]

Headings at a place are unique and only assigned to paths the place is on. Formally,

$$radial(p, pa, dir, h) \wedge radial(p, pa, dir, h') \rightarrow h = h' \ , \tag{7.8}$$

$$radial(p, pa, dir, h) \rightarrow on(pa, p) \ . \tag{7.9}$$

We introduce the predicate

$$angle(p, pa, dir, pa1, dir1, ang)$$

to represent the angle the agent will have to turn to face path *pa1* in direction *dir1* when it is at place *p* facing path *pa* in direction *dir*. Angles take values in $[0, 2\pi)$. The predicate *angle* is defined as follows:

$$\begin{aligned} & angle(p, pa, dir, pa1, dir1, ang) \equiv \\ & \quad \exists h_{pa},\ h_{pa1}\ \{radial(p, pa, dir, h_{pa}) \wedge radial(p, pa1, dir1, h_{pa1}) \wedge \\ & \qquad ang \in [0, 2\pi) \wedge \\ & \qquad ang - (h_{pa1} - h_{pa}) \ = \ 0\ mod\ 2\pi\} \ . \end{aligned} \tag{7.10}$$

We use the predicate

$$angle^{\approx}(p, pa, dir, pa1, dir1, I_{ang})$$

to denote the fact that I_{ang} is an estimate of the angle at place *p* between path *pa* in direction *dir* and path *pa1* in direction *dir1*. Estimates of the angle between paths at a place are gathered from *turn* actions. Angle estimates are compounded and merged as we did for distances among places in a path:

$$\{\ min\ angle^{\approx} :$$

$$\begin{aligned} & [\langle\ ds, (turn\ I_{ang}), ds'\ \rangle \wedge at(ds, p) \wedge along(ds, pa, dir) \wedge along(ds', pa1, dir1)] \\ & \quad \rightarrow angle^{\approx}(p, pa, dir, pa1, dir, I_{ang}) \ , \end{aligned} \tag{7.11}$$

$$\begin{aligned} & [angle^{\approx}(p, pa, dir, pa1, dir1, I_{ang}) \wedge angle^{\approx}(p, pa1, dir1, pa2, dir2, I_{ang'})] \\ & \quad \rightarrow angle^{\approx}(p, pa, dir, pa2, dir2, I_{ang} + I_{ang'}) \end{aligned} \tag{7.12}$$

$$\}$$

$$angle^{\otimes}(p, pa, dir, pa1, dir1, I) \equiv_{def} I = \cap\{I_{est}\ :\ angle^{\approx}(p, pa, dir, pa1, dir1, I_{est})\} \tag{7.13}$$

Finally, the angle between paths at a place has to be *compatible* with all of its estimates. Formally,

$$angle^{\otimes}(p, pa, dir, pa1, dir1, I_{ang}) \rightarrow \exists ang \in I_{ang}\ angle(p, pa, dir, pa1, dir1, ang) \tag{7.14}$$

7.2.3 Two dimensional frames of reference

Notice that the SSH does not explicitly represent the distance or direction between two arbitrary places. In order to do so, distances between places on a path as well as the angles between paths at a place must be combined. We use the predicate

$$location2(p, q, l)$$

to indicate that the location of place q with respect to the two dimensional frame of reference associated with place p is l (a real valued pair). The location of a place w.r.t. a frame of reference is unique,

$$location2(p, q, l) \wedge location2(p, q, l') \rightarrow l = l' \ . \tag{7.15}$$

We do not restrict what places are assigned locations with respect to a given two dimensional frame of reference. Assigning locations to places in a common frame of reference is a costly operation. In practice, not all places occur in all frames of reference nor does a frame of reference include all places in the map.

When restricted to environments with "straight" paths,[9] it is possible to state when a two dimensional frame of reference is *compatible* with the actual experiences of the robot. The next axioms state this requirement:

$$\begin{aligned} & location2(p, p1, l_{p1}) \wedge location2(p, p2, l_{p2}) \wedge path\_distance^{\otimes}(pa, p1, p2, I_d) \\ & \quad \rightarrow |l_{p1} - l_{p2}| \in I_d \ . \end{aligned} \tag{7.16}$$

$$\begin{aligned} & [location2(p, p1, l_{p1}) \wedge location2(p, p2, l_{p2}) \wedge location2(p, p3, l_{p3}) \wedge \\ & \quad order(pa, dir, p1, p2) \wedge order(pa', dir', p2, p3) \wedge angle^{\otimes}(p2, pa, dir, pa', dir', I_{ang})] \\ & \quad \rightarrow angle(-\vec{l_{p2}l_{p1}}, \vec{l_{p2}l_{p3}}) \in I_{ang} \ , \end{aligned} \tag{7.17}$$

where $angle(\vec{v}, \vec{w})$ denotes the angle in $[0, 2\pi)$ from vector $\vec{v}$ to vector $\vec{w}$.

Axioms 7.16 and 7.17 assume that paths are straight. In order to deal with more general paths, one should include some parameters describing the shape of the path, or at least an estimate of the change in heading while traveling.[10] This last approach was adopted in [Kuipers and Levitt, 1988] where travel actions were represented as $\langle ds, (travel \ \ dist \ \ \triangle\theta), ds' \rangle$, where $dist$ corresponds to the distance between the places associated with ds and ds', and

[9]Environments where the control law trajectories are straight lines.

[10]There is a deeper semantic issue here. When curved paths are possible, the predicate $path\_distance$ represents distance *along the path*, not straight-line distance between end point. To handle curved paths, we have to separate those two concepts, or have estimates of both types of "distances".

$\triangle\theta$ corresponds to the change of orientation while traveling. However, there is not a statement of how this extra information is used or whether it suffices to describe appropriate metrical constraints for two dimensional frames of reference. While a more detailed account of the use of metrical information is desirable, including representing and reasoning about a path's shape, we have left this description outside the scope of this work. While restricted to straight paths, our current presentation of the SSH metrical level illustrates the main considerations in order to represent and use metrical information.

7.3 Combining topological and metrical information

While radial and one dimensional frames of reference are associated with any place and path, respectively, there is not a general SSH theory asserting when to create a two dimensional frame of reference, what places should be included in a such frame of reference, or how to assign place locations consistent with the estimates of distances and angles gathered by the agent. Nevertheless, in section 7.4 we address these issues from the SSH implementation point of view.

In this section we formally state what it means for the topological map to be consistent with a given set of frames of reference. In order to do so, given distinctive states $ds, ds_1, \ldots, ds_n$, we introduce the notation $\langle ds \ : \ ds_1, \ldots, ds_n \rangle$ to state that in the two dimensional frame of reference associated with ds's place, the places associated with the different ds_i have a location.

Definition 3

Let $ds, ds_1, \ldots, ds_n$ be a set of distinctive states. By definition,

$$\begin{aligned} &\langle ds \ : \ ds_1, \ldots, ds_n \rangle \ \equiv_{def} \\ &\qquad \exists p \ \left\{ at(ds, p) \wedge \bigwedge_{i=1}^{n} \exists p_i, l_i \ [at(ds_i, p_i) \wedge location2(p, p_i, l_i)] \right\} \end{aligned}$$

$\{$*end of definition*$\}$

By $\mathbf{2D\_Frames}$ we denote the formula specifying any two dimensional frames of reference used by the agent. Without loss of generality, we require two dimensional frames of reference to be specified as in definition 3. We require any model of the SSH to have only the two dimensional frames of reference specified in $2D\_Frames$. In addition, the places belonging to a frame of reference should be only those explicitly stated in 7.18. These last two requirements can be stated as follows:

$$\{\ min\ location2\ :\ 2D\_Frames\ \} \tag{7.18}$$

The block **AT_block** (see page 41) is updated by adding axioms 7.1 to 7.18 so that when determining places and paths, those should conform to the metrical constraints in E. Below we present the updated version of this block:[11]

$AT\_block =$

$\{\ max\ teq\ :$

Axioms 7.1-7.18,

$Axioms\ \ 6.1 - 6.12,$

$teq\ is\ an\ equivalence\ relation\ ,$

$teq(ds_1, ds_2) \rightarrow View(ds_1, v) \equiv View(ds_2, v),$

$teq(ds_1, ds_2) \wedge \langle\ ds_1, a, ds'_1\ \rangle \wedge \langle\ ds_2, a, ds'_2\ \rangle \rightarrow teq(ds'_1, ds'_2),$

$teq(ds_1, ds_2) \rightarrow \forall p\, [at(ds_1, p) \equiv at(ds_2, p)] \wedge$

$\qquad\qquad \forall pa, dir\, [along(ds_1, pa, dir) \equiv along(ds_2, pa, dir)]\ \ ,$

$\langle ds, turn, ds' \rangle \rightarrow \neg teq(ds, ds')\ ,$

$\langle ds, turnAround, ds' \rangle \wedge \langle ds, TurnAround, ds'' \rangle \rightarrow teq(ds', ds'')\ ,$

$\langle\ ds_1, turnAround, ds_2\ \rangle \wedge \langle\ ds_2, turnAround, ds_3\ \rangle \rightarrow teq(ds_1, ds_3)\ ,$

$at(ds, p) \rightarrow tplace(p),$

$\exists! p\, at(ds, p),$

$turn\_eq(ds_1, ds_2) \equiv \forall p\, [at(ds_1, p) \equiv at(ds_2, p)]\, ,$

$\{min\ \ turn\_eq :$

$teq(ds_1, ds_2) \wedge teq(ds_3, ds_4) \wedge \widehat{turn}(ds_2, ds_3) \rightarrow turn\_eq(ds_1, ds_4),$

$turn\_eq(ds_1, ds_2) \wedge turn\_eq(ds_2, ds_3) \rightarrow turn\_eq(ds_1, ds_3)$

$\}$

$along(ds, pa, dir) \rightarrow tpath(pa),$

$\{\ min\ along :$

$\langle\ ds, travel, ds'\ \rangle \rightarrow \exists pa,\ dir\ \left[along(ds, pa, dir) \wedge along(ds', pa, dir)\right]\, ,$

$\langle ds, turnAround, ds' \rangle \rightarrow along(ds, pa, dir) \equiv along(ds', pa, -dir),$

[11] We indicate the additions in bold letters.

$$
\begin{array}{l}
\quad teq(ds_1, ds_2) \rightarrow along(ds_1, pa, dir) \equiv along(ds_2, pa, dir) \\
\} \\
along(ds, pa, dir) \wedge along(ds, pa1, dir1) \rightarrow pa = pa1 \wedge dir = dir1, \\
at(ds_1, p) \wedge at(ds_2, p) \wedge along(ds_1, pa, dir) \wedge along(ds_2, pa, dir) \rightarrow teq(ds_1, ds_2), \\
\\
[\langle ds, turn\_desc, ds' \rangle \;\wedge\; turn\_desc \neq turnAround \wedge \\
\qquad along(ds, pa, dir) \wedge along(ds', pa1, dir1)] \rightarrow pa \neq pa1, \\
\\
\{\; min\ order: \\
\quad [\langle\, ds, travel, ds' \,\rangle \wedge at(ds, p) \wedge at(ds', q) \wedge \\
\qquad along(ds, pa, dir) \wedge along(ds', pa, dir)] \rightarrow order(pa, dir, p, q), \\
\quad order(pa, pos, p, q) \equiv order(pa, neg, q, p), \\
\quad order(pa, dir, p, q) \wedge order(pa, dir, q, r) \rightarrow order(pa, dir, p, r) \\
\} \\
\\
\neg order(pa, dir, p, p), \\
\{min\ on:\; at(ds, p) \wedge along(ds, pa, dir) \rightarrow on(pa, p)\ \} \\
on(pa, p) \wedge on(pa, q) \wedge tpath(pa) \rightarrow \\
\qquad \exists ds_1, dir_1, ds_2, dir_2\ [at(ds_1, p) \wedge along(ds_1, pa, dir_1) \wedge at(ds_2, q) \wedge \\
\qquad\qquad\qquad\qquad along(ds_2, pa, dir_2) \wedge travel\_eq(ds_1, ds_2)]\,, \\
\{min\;\; travel\_eq: \\
\quad \widehat{travel}(ds_1, ds_2) \rightarrow travel\_eq(ds_1, ds_2), \\
\quad \langle\, ds_1, turnAround, ds_2 \,\rangle \rightarrow travel\_eq(ds_1, ds_2) \wedge travel\_eq(ds_2, ds_1), \\
\quad teq(ds_1, ds_2) \wedge teq(ds_3, ds_4) \wedge travel\_eq(ds_2, ds_3) \rightarrow travel\_eq(ds_1, ds_4), \\
\quad travel\_eq(ds_1, ds_2) \wedge travel\_eq(ds_2, ds_3) \rightarrow travel\_eq(ds_1, ds_3) \\
\} \\
\\
\mathbf{circ}\;\; tpath \succ \neg in\_region \succ tplace\ \mathbf{var}\;\; new\vec{SSH}pred
\end{array}
$$

}

where $new\vec{SSH}pred$ stands for the tuple of predicates

$$\langle\;\; at,\ along,\ order,\ on,\ teq,\ turn\_eq,\ travel\_eq,$$

$$totheRightOf,\ totheLeftOf,\ leftOf,\ rightOf,\ is\_region,$$
$$\mathbf{radial}, \mathbf{position1}, \mathbf{position2},$$
$$\mathbf{path\_distance}, \mathbf{path\_distance}^{\approx}, \mathbf{path\_distance}^{\otimes},$$
$$\mathbf{angle}, \mathbf{angle}^{\approx}, \mathbf{angle}^{\otimes}$$
$$\rangle\ .$$

While the priorities in the circumscription policy are kept as before, by adding axioms 7.1-7.18 inside the circumscription scope, we require the topological map to have a minimum set of paths and places consistent with the metrical information. The next examples illustrate how metrical information is used to disambiguate the topological map.

Example 25

Consider example 19 (page 59) where two topological maps are consistent with the agent's experiences (see figure 7.1). Suppose that "perfect" metrical information is available to the agent such that the following set of schemas are created by the agent:[12]

$$\begin{array}{ll}
\langle ds1, (turn\ \ -90^o), ds2\rangle & \langle ds2, (travel\ \ 10), ds3\rangle \\
\langle ds3, (turn\ \ 90^o), ds4\rangle & \langle ds4, (turn\ \ -90^o), ds3\rangle \\
\langle ds3, (travel\ \ 10), ds5\rangle & \langle ds5, (turn\ \ -90^o), ds6\rangle \\
\langle ds6, (travel\ \ 20), ds7\rangle & \langle ds7, (turn\ \ -90^o), ds8\rangle \\
\langle ds8, (travel\ \ 10), ds9\rangle & \langle ds9, (turn\ \ -90^o), ds10\rangle \\
\langle ds10, (travel\ \ 20), ds11\rangle &
\end{array}$$

How does the agent figure out that it is back to $ds4$ rather than to $ds1$?. As claimed in example 19 (page 59) both options $teq(ds4, ds11)$ and $teq(ds1, ds11)$ are topologically possible (figures 7.1b,c). However, given the metrical information above, only the assumption $teq(ds4, ds11)$ is a consistent one. To deduce this fact, the agent includes the frame of reference $\langle ds4\ :\ ds1, \ldots, ds11\rangle$ in E, which renders impossible $teq(ds4, ds11)$.

Should the metrical information have been less precise, the agent might not benefit from this extra metrical information. For example, suppose that instead of sharp 90^o turn angles, there exists a $\pm 10^o$ uncertainty associated with the turn actions above (i.e. consider replacing $\langle ds1, (turn\ \ -90^o), ds2\rangle$ by $\langle ds1, (turn\ \ [-110^o, -80^o], ds2\rangle$). In this case the agent cannot use metrical information to deduce that it is back to $ds4$ and it will have two topological maps consistent with its information.
{*end of example*}

[12]Whenever we use a number x instead of an interval, it is an abbreviation for $[x, x]$.

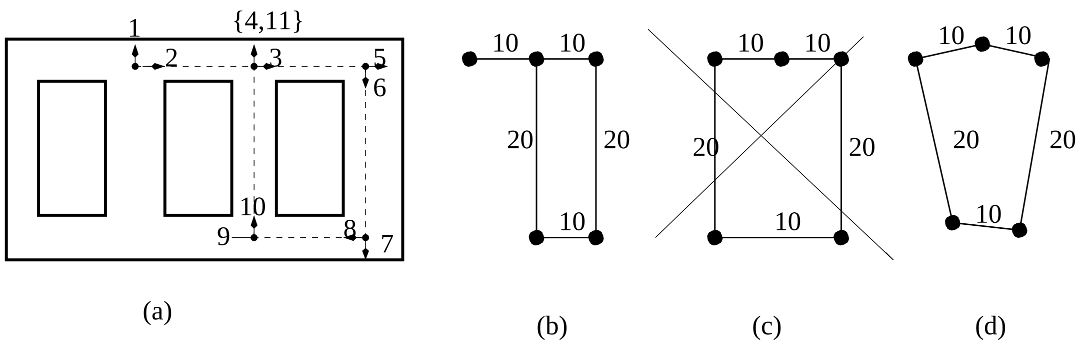


Figure 7.1: (a) The robot goes around the block visiting distinctive states $ds1$ to $ds11$ in the order suggested by the figure. Distinctive state $ds11$ is observed at the same environment state as $ds4$. Assume distinctive states $ds1$ and $ds4$ look alike to the agent.(b) and (c) represent two possible topological maps for the environment in (a) (see example 19, page 59). The model in (c) can be discarded as it is not consistent with the available metrical information. (d) With $\pm 10^o$ noise associated with turn actions, the agent cannot use metrical information to discard the environment depicted in (c).

The example above may suggest that metrical information is used to check whether an already built topological map is consistent with metrical information. However, by including axioms 7.1-7.18 inside AT_block, metrical information is used while building the topological map. As the next example illustrates, this may imply that the agent identifies more places than it does when not using metrical information.

Example 26

Consider example 24 where three places and three paths where identified while the agent explored a square room (figure 7.2b). Suppose the agent has access to perfect metrical information and uses it while building the metrical map. In this case, the set of schemas associated with this environment will be:

$$\begin{array}{ll}
\langle ds1, (turn\ -90^o), ds2\rangle & \langle ds2, (travel\ 10), ds3\rangle \\
\langle ds3, (turn\ -90^o), ds4\rangle & \langle ds4, (travel\ 10), ds5\rangle \\
\langle ds5, (turn\ -90^o), ds6\rangle & \langle ds6, (travel\ 10), ds7\rangle
\end{array}$$

In order to decide whether the agent is back to $ds1$, the frame $\langle ds1 \ : \ ds1, \ldots, ds7\rangle$ is created. It is easy to see that it is not possible to have $teq(ds1, ds7)$ while satisfying the metrical constraints. Consequently, the topological map will have *four* places instead of *three*, as illustrated in figure 7.2c.
{*end of example*}

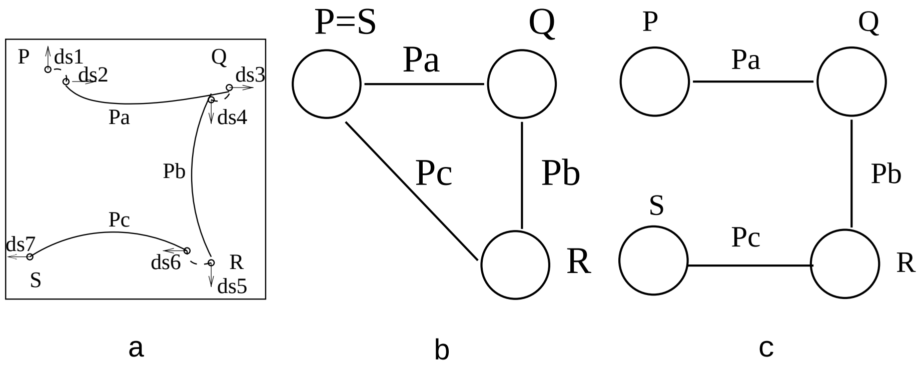


Figure 7.2: (a) The agent visits distinctive states $ds1$ to $ds7$ by the order suggested in the figure. Suppose all corners look alike to the agent. In particular, $ds1$ and $ds7$ share the same view. (b) Topological map associated with (a) when metrical information is not available. (c) Topological map associated with (a) when metrical information is available. In this case, the places associated with $ds1$ and $ds7$ are different ($P \neq S$).

While in the examples above all visited distinctive states were included in a two dimensional frame of reference, this is in general not the case. In the presence of metrical uncertainty, a global frame of reference may not provide useful information to determine whether two places are the same, or to estimate the distance between two arbitrary places (example 25).

7.4 Creating two dimensional frames of reference

When using two dimensional frames of reference, the SSH does not say when a frame of reference should be created, or what places should be included in a given frame of reference. Nor does the SSH say how to assign locations to places in a given frame of reference. In this section we explore different answers to these issues.

7.4.1 When to create two dimensional frames of reference

In order to answer when one should create a metrical frame of reference, one should consider the related question, why to use frames of reference. In examples 25 and 26 frames of reference were created in order to decide whether two distinctive states are the same. Another application of two dimensional frames of reference is to determine the relative orientation and distance between arbitrary places in order to, for example, find new routes or orient in the environment. In this last case, a place is included in a frame of reference as long as its location uncertainty is appropriate for drawing useful conclusions. Next we illustrate these possibilities.

Distinguishing distinctive states

Suppose the agent is at distinctive state ds which shares the same view as distinctive states ds'. Moreover, suppose that it is topologically consistent to assume that ds and ds' denote the same environment state. In order to check if ds and ds' could be the same environment state, the agent finds the shortest path from ds' to ds, $ds' = ds_0, \ldots, ds_n = ds$, and creates a frame of reference $\langle ds' \quad : \quad ds_0, ds_1, \ldots, ds_n \rangle$.[13] The effect of having this frame of reference when building the topological map is such that $teq(ds, ds')$ will be the case if there exists at least one location assignment to the places in the frame of reference that is consistent with the metrical constraints. Though the method is only partially conclusive, it helps to distinguish places before attempting more computationally expensive methods (e.g. the rehearsal procedure).

Patchwork mapping

Having a global frame of reference including all places in the map is usually inappropriate since the uncertainty associated with some places' locations in such a frame of reference may not allow the agent to draw useful conclusions. Instead, the agent can have multiple frames of reference as well as relations among the different frames of reference [McDermott and Davis, 1984, Kuipers, 2000]. As the agent explores the environment, new frames of reference are created when the current's location uncertainty with respect to the current frame of reference is larger than a given threshold [Moutarlier and Chatila, 1989, Engelson and McDermott, 1992b].

7.4.2 How to create two dimensional frames of reference

The problem of assigning locations to places given some metrical constraints can be solved by borrowing methods from different fields. For example, estimation theory tells us how to estimate the true value of a given set of variables given noisy observations of the relations between those variables [Gelb, 1974, Smith and Cheeseman, 1986]. The robotics community has developed algorithms to solve a network of spatial relations [Durrant-Whyte, 1987, Durrant-Whyte, 1988a, Durrant-Whyte, 1988b, Moutarlier and Chatila, 1989]. Techniques from multidimensional scaling [Borg and Groenen, 1997] and nonlinear programming [Peressini *et al.*, 1988] can also be used.

[13]The shortest path with respect to the number of edges in the path. The heuristic of choosing the shortest path assumes that the fewer places whose location must be determined, the more accurately their locations can be determined.

Next we show an algorithm to create a two dimensional frame of reference while preserving metrical constraints. The algorithm borrows its main ideas from a constraint propagation algorithm presented in [Hernandez, 1994].

7.4.3 "Mental" route rehearsal

The proposed method to integrate multiple frames of reference is based on the idea that the SSH supports "mental route rehearsal". That is, the agent can "imagine himself" following a route and assigning locations to the different places in the route. Information from multiple routes is combined to further constrain the assignment of locations to the different places. Next, we make these ideas precise.

A *route* is a sequence of the form $P_0\,(S_0, d_0)\,P_1\,(S_1, d_1), \ldots P_n$, where S_i is a topological path connecting places P_i and P_{i+1} when traveling in direction d_i (i.e. $order(S_i, d_i, P_i, P_{i+1})$ is the case). In addition, we require that all the places in the route are different, except possibly P_0 and P_n.[14]

The problem we are to solve is to map each place in a set of routes to a "location" in R^2. These locations are such that they preserve the estimated distance and relative orientation between consecutive places in the route. In this work, a *location* is a two dimensional rectangle whose sides are aligned with the coordinate axes of R^2.[15] In order to assign locations to places in a route, an initial location L_0, an initial heading H_0, and a vector V_0 must be specified. L_0 is the location assigned to P_0. By mapping the local heading H_0 at place P_0 to the vector V_0, the agent defines the orientation relationship between the local radial frame of reference at P_0 and R^2.

Once L_0, H_0, and V_0 have been provided, the location of the other places in the route is calculated by starting at P_0 with location L_0 and integrating distances between places and change of heading among the paths in the route R.[16] As a result of "following the route", each place P_i in the route has associated a location $Loc(P_i, R, L_0, H_0, V_0)$. By convenience, we will write $Loc(P_i, R)$ instead of $Loc(P_i, R, L_0, H_0, V_0)$, whenever L_0, H_0 and V_0 are understood.

[14] $P_i = P_j \wedge i < j \rightarrow i = 0 \wedge j = n$.

[15] This rectangle accounts for the uncertainty in the SSH's metrical information. If there is no uncertainty, then the rectangle reduces to a point in R^2.

[16] The change of heading at place p from path $s1$ to path $s2$ is the angle an agent will have to rotate once it gets into place p following path $s1$, and leaves p following path $s2$.

Next we introduce the notation and definitions we use to assign locations to places. Definition (6) specifies how to assign locations to places in a given route.

Definition 4 (Rect(A))

Given two set of vectors in R^2, A and B, $A + B$ denotes the set of vectors $\{a + b : a \in A, b \in B\}$.

Given a set of vectors, A, **Rect(A)** denotes the minimum rectangle containing the set A, such that the sides of Rect(A) are aligned with the coordinate axes of R^2.

Given a vector v in R^2 and an angle $\theta \in [0, 2\pi)$, v_θ denotes the unit vector associated with the rotation of v by an angle of θ.
$\{$*end of definition*$\}$

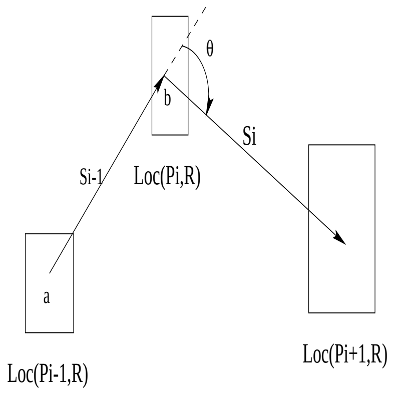


Figure 7.3: Calculating the location of P_{i+1} given the location of P_i and P_{i-1} (see text). θ is the change of heading from path S_{i-1} to path S_i at place P_i.

For a route R the operator $\oplus_R$ defines how to calculate the location of place P_{i+1} given the location of places P_{i-1} and P_i.[17]

Definition 5 ($\oplus_R$)

Given a route R, $R = P_0\, S_0\, P_1 \ldots S_{n-1}\, P_n$, an initial heading H_0, an initial vector V_0, an initial location L_0, and locations L_{i-1} and L_i for places P_{i-1} and P_i, respectively, $\oplus_R$ is

[17]Special cases will be considered for the locations of P_0 and P_1.

defined as follows (see Figure 7.3):

$$
\begin{aligned}
&(V_0,\, H_0) \;\oplus_R\; (P_0,\, L_0) \;= \\
&\quad Rect(\{b \,+\, \lambda V_{0\theta} \,:\, b \in L_0,\ \lambda \in I_d,\ path\_distance^{\otimes}(S_0, P_0, P_1, I_d), \\
&\qquad\qquad \theta \,=\, H_0 \,-\, h,\ \ radial(P_0, S_0, d_0, h)\})
\end{aligned}
$$

$$
\begin{aligned}
&(P_{i-1},\, L_{i-1}) \;\oplus_R\; (P_i,\, L_i) \;= \\
&\quad Rect(\{b \,+\, \lambda(\frac{b-a}{|b-a|})_\theta \,:\, a \in L_{i-1}\ ,\ b \in L_i,\ a \neq b, \\
&\qquad\qquad \lambda \in I_d,\ path\_distance^{\otimes}(S_i, P_i, P_{i+1}, I_d), \\
&\qquad\qquad \theta \in I_{ang},\ \ angle^{\otimes}(P_i, S_{i-1}, d_{i-1}, S_i, d_i, I_{ang})\ \})
\end{aligned}
$$

{*end of definition*}

In order to assign locations to the places in a route one starts with P_0 in L_0 and applies $\oplus_R$ recursively until finding a location for P_n. Definition 6 illustrates this case. It is also possible to use $\oplus_R$ to combine locations along multiple routes as defined in the next section.

Definition 6 (Route Vector Addition)

Given a route R, $R \,=\, P_0\, S_0\, P_1 \ldots S_{n-1}\, P_n$, an initial heading H_0, an initial vector V_0, and an initial location L_0, we define $Loc(P_i, R)$, the location of P_i according to route R, as follows:

1. If $P_i = P_n = P_0$ then $Loc(P_n, R)$ = $Loc(P_0, R)$ = L_0.
2. If $P_i = P_1$ then $Loc(P_1, R) = (V_0, H_0) \;\oplus_R\; (P_0, L_0)$.
3. Otherwise, $Loc(P_i, R) = (P_{i-2},\, Loc(P_{i-2}, R)) \;\oplus_R\; (P_{i-1},\, Loc(P_{i-1}, R))$

{*end of definition*}

7.4.4 Combining information from multiple routes

Information from multiple routes can be used to further constrain the location of a place P with respect to a reference place P_0. For example, in figure 7.4 information from two different routes $R1$ and $R2$ further constrain the location of places D and E with respect to

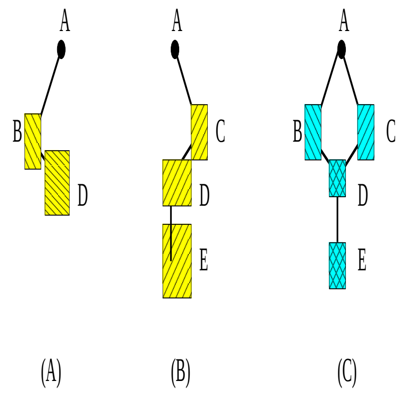


Figure 7.4: (a) Route $R1$ and the associated locations for places B and D. Location of places is indicated by a rectangle. (b) Route $R2$ and the associated locations for places C, D and E. (c) The location of place D is changed by intersecting its locations with respect to routes $R1$ and $R2$. The new location is then propagated to update E's location.

place A.

A constraint propagation algorithm is proposed to mix information from various routes. The basic idea of this algorithm is as follows. Given a set of routes SR, we associate a directed graph G with SR. The nodes of G are the places in SR. An edge (P,Q) belongs to G if there exists a route in SR such that Q is immediately after P. Whenever the location of a place P is changed, this location is used to update the location of the neighbor places of P according to G. A place location is changed by intersecting its previous location with a new one. The algorithm ends when there is no place whose location must be changed.

Let SR = $\{R_1, \ldots, R_n\}$ be a set of routes, all of which start at the same place P_0. Let $Places(SR)$ denote the set of places belonging to the routes in SR. Given a place P in $Places(SR)$, we are to define $Loc(P, SR)$, the location of P given the routes in SR. As explained above, in order to associate locations to places, we have to specify an initial heading H_0, an initial vector V_0, and a location L_0. $Loc(P, SR)$ depends on these parameters. Next we describe how the propagation algorithm works.

Initially the location of all the places, except P_0, is set to R^2 (since R^2 is the identity for the intersection of locations). The location of P_0 is set to L_0.

```
Procedure **Initialization**
    empty(Queue);
    for each place p ∈ Places(SR) do
        $Loc(p, SR)$ = $R^2$;
    $Loc(P_0, SR)$ = $L_0$;
    push($P_0$,$Queue$)
{*end of procedure*}
```

The algorithm keeps a priority queue *Queue* where nodes whose location has changed is kept. We will explain later how places are added to *Queue*. To calculate the location of the different places in SR, the procedure *compute closure* is called.

```
Procedure **Compute Closure**
    while $Queue \neq \emptyset$ do
        begin
            p = pop($Queue$);
            Propagate(p);
        end
{*end of procedure*}
```

Whenever the location of a place is changed, its new location is propagated to its neighbor places. We use the predicate $Next(SR, R, p, q)$ to denote the fact that R is in SR, and place q is immediately after place p in route R. Given a new location for p, we define the new location of q to be the intersection of its current location according to SR (i.e. $Loc(q, SR)$) and the location of q according to R given the new location of p (as defined by the operator $\oplus_R$, definition 5). Since the new location of a place is always a subset of its previous location, we decide whether the location of q has changed by comparing the area of its previous and new location (see definition 8, page 110[18]).

[18]We consider special cases when a location is a "degenerate" rectangle, like a line segment or a point.

```
Procedure **Propagate (p)**
    for each place $q$ such that $Next(SR, R, p, q)$ do
        begin
            if $\exists p',\ (p', p, R)$
                then New := $Loc(q, SR) \cap (p',\ Loc(r, SR) \oplus_R (p,\ Loc(p, SR))$
                else New := $Loc(q, SR) \cap (V_0, H_0) \oplus_R (p,\ Loc(p, SR))$
            if New = $\emptyset$
            then signal contradiction
            if Change(New, $Loc(q, SR)$)
                then
                    begin
                        add q to $Queue$
                        $Loc(q, SR)$ := New
                    end
        end
{end of procedure}
```

The next example illustrates how the algorithm works to combine the information from the routes in figure 7.4.

Example 27

Consider the routes $R1$ and $R2$ depicted in figure 7.4. We start the algorithm by calling the procedure initialization, after which we have that[19]

```
Loc(A) = {(0,0)}, Loc(B) = Loc(C) = Loc(D) = Loc(E) = R^2
Queue = [A]
```

The procedure Compute Closure is then invoked, A is removed from *Queue*, and the procedure $Propagate(A)$ is called. New locations for places B and C are then calculated. The new locations become:

```
Loc(A) = {(0,0)}, Loc(B) = L1, Loc (C) = L2,  Loc(D)=Loc(E) = R^2
Queue = [B,C]
```

where $L1$ and $L2$ are the locations calculated for B and C respectively. Now B is taken from *Queue* and $Propagate(B)$ is called, after which we have that

```
Loc(A) = {(0,0)}, Loc(B)= L1, Loc (C)= L2, Loc(D)= L3, Loc(E) = R^2
Queue = [C,D]
```

$Propagate(C)$ will be called and the location of D is changed by intersecting information from both routes,

[19]We will omit the parameter SR in $Loc(p, SR)$ and write $Loc(p)$.

```
Loc(A) = {(0,0)}, Loc(B) = L1, Loc (C) = L2,  Loc(D)= L3 inter L3'
Loc(E) = R^2
Queue = [D]
```

L3 and L3' are the location of D according to route $R1$ and $R2$, respectively. Finally, $Propagate(D)$ is called which will give a value for the location of E.
{*end of example*}

In the example above, we did not propagate the location of any place twice. In general this is not the case, and we need to be more strict on how places are taken from $Queue$ when $Propagate$ is called. In order to do so, we impose an order, $before_{SR}(p, q)$, among the places in SR. Next we define $before_{SR}(p, q)$.

Definition 7 ($before_{SR}(p, q)$**)**

Given a set of routes SR, places p and q in $Places(SR)$, we say that place p is before place q according to SR, $before_{SR}(p, q)$, if (i) there exists a route R in SR such that p is before q in R, and (ii) there does not exist a route R in SR such that q is before p in R.
{*end of definition*}

The relation $before_{SR}$ defines a partial order among the places in SR. We use it to prioritize the places in Queue, such that whenever $before_{SR}(p, q)$ is the case, the priority of p is greater than the priority of q.

In order to decide whether the location of a place has changed, we compare the areas between the new and the old place location, as stated in the next definition.

Definition 8 (Change(new,old))

Given two locations, new and old, such that $new \subseteq old$, **change(new,old)** is true if and only if one of the following conditions is true

1. old = R^2 and new $\neq R^2$.
2. both old and new are rectangles and $[area(old) - area(new)] > \epsilon$, for a given $\epsilon > 0$.
3. old is a rectangle and new is a line segment or a point.
4. both old and new are line segments and $[length(old) - length(new)] > \epsilon$, for a given $\epsilon > 0$.

5. old is not a point and new is a point.

{*end of definition*}

The proposed algorithm assigns locations to places in a "liberal" way.[20] By choosing an arbitrary point from a place's location as the actual place position, the resulting assignment of positions does not necessarily satisfy the distance and angle constraints among places. However, any assignment of places' positions satisfying these constraints is covered by the places' locations produced by our algorithm. Should the application at hand require such assignment of positions, starting from the locations assigned by our algorithm one could use Monte Carlo or hill-climbing algorithms to find such assignment (if exists) [McDermott and Davis, 1984].

Possible applications of our algorithm include answering queries about the spatial layout and visualization of the topological map:

- Answering some queries about the spatial layout can be rendered to check some properties in the location assigned to places. For example, to answer whether place A is to the left or right of place B, when facing heading θ at B, the following procedure is used. Find a route from B to A. Let P_0 be B, $H_0 = \theta$ and $V_0 = (0, 1)$. A is to the left of B if the location of A is contained in $\{(x, y) : x < 0)\}$. A is to the right of B if the location of A is contained in $\{(x, y) : x > 0)\}$. Otherwise the agent does not decide whether A is to the left or right of B. Notice that given the uncertainty associated with distances and angles, the proposed method is partially conclusive: it could answer *yes*, *no* or *maybe*.

- Displaying the layout of places in a computer screen is important for a robot-human interface. Having pre-defined two dimensional frames of reference allows one to quickly display a region or neighborhood, a display that is useful for specifying navigation goals as well as generating explanations of the robot's navigation plans.

7.5 Summary

In this chapter we have defined different kinds of frames of reference considered in the SSH: one dimensional frames of reference which assign positions to places on a path, radial frames of reference that assign angles to the different paths that intersect at a given

[20]We borrow this term from [McDermott and Davis, 1984]. In a "liberal" map, the location of places grants as much freedom as possible to each place even though not every coordinates we pick within a place location will satisfy all metrical constraints. This would be the case in a "conservative" map.

place, and two dimensional frames of reference which assign positions to arbitrary set of places. Our formalization describes how uncertainty associated with metrical information is taken into account when creating these frames of reference. In particular, we have described how to represent different metrical measurement estimates, how to combine them, and how to represent the constraints they impose on the metrical relations among different objects referenced by a frame of reference.

Uncertainty in metrical information has been represented by one and two dimensional rectangles. This representation is adequate for most practical uses. However, more sophisticated representations of uncertainty are compatible with our methods. For example, probability density functions (such as Gaussians) can be used to represent uncertainty on the distances between places in a path.

The SSH theory does not specify when to create two dimensional frames of reference. We have described particular instances when this should be done: in order to disambiguate places or when uncertainty in a place location is "high". Given that the actual formal treatment of uncertainty propagation is outside the scope of this work, we have provided pointers to the relevant fields where such studies are done (see section 7.4.1). Nevertheless, we have described a constraint propagation algorithm to integrate multiple frames of reference while assigning locations to places in a given set of routes. Locations are assigned such that they preserve the estimated distance and relative orientation between consecutive places in a path (see section 7.4.3).

Finally, we have pointed out that our current description of two dimensional frames of reference assumes environments where paths are straight. Nevertheless, our current presentation of the SSH metrical level illustrates the main considerations in order to represent and use metrical information. A more detailed account of the use of metrical information, possibly by reasoning about the shape of paths, is suggested as future work.

Chapter 8

Regions

As the agent explores its environment, the number of places it identifies grows. Consequently, the topological map, as described until now, might have so much information that it will be cumbersome (i.e. time-consuming) to use for future navigation or for a human level interface.[1] It is useful then to define a hierarchical representation that simplifies some details in the topological map. The basic abstraction method we use is to group $places$ into $regions$. As for topological places, regions are ordered along routes, and in turn they can be grouped to form new regions, giving the representation a powerful mechanism of abstraction. Metrical information about regions, like their distance along a route or their relative direction, can be derived from the corresponding metrical relations among the places in the regions. In this chapter we state how to represent regions in the SSH as well as the properties of the hierarchical representation associated with them. We also illustrate the use of regions for hierarchical planning and qualitative reasoning about space.[2] Finally, we also provide some criteria for automatically identifying regions and creating their associated hierarchy.

8.1 Including regions in the SSH topological level

$Regions$ are sets of places. $Regions$ themselves can be grouped to form new regions. Since regions can be seen as places, we have not added a new sort $regions$ to the SSH

[1]Similar problem occurs when a QSIM behavior graph has many states. A way to simplify these behaviors graphs is to abstract together adjacent states that, by according to a user's criteria, need not be distinguished [Mallory *et al.*, 1996].

[2]Further uses of regions can be found in section A.3, page 209 (hierarchical plan execution).

topological level ontology. We use the predicate[3]

$$in\_region(p, r)$$

to denote that *place* p is in *region* r. Recall that the sort of places contains the subsorts of topological paths and regions. In chapter 5 we required that (see axiom 5.4, page 39)

$$\neg\exists p\,[tplace(p) \wedge is\_region(p)]$$

where p is a variable taking values in the sort of *places*. Similarly, in chapter 6 we restricted the domain of *in_region* such that (see axiom 6.6, page 80)

$$in\_region(p, r) \rightarrow is\_region(r)\ .$$

We require that the transitive closure of *in_region* defines a DAG[4]. Formally, we restrict our attention to models of the following formulae:

$$\begin{array}{l} \neg in\_region^*(r, r), \\ \{\ min\ in\_region^*\ \ : \\ \ \ in\_region(p, r) \rightarrow in\_region^*(p, r), \\ \ \ in\_region^*(p, q) \wedge in\_region^*(q, r) \rightarrow in\_region^*(p, r) \\ \} \end{array}$$

where in_region^* denotes the transitive closure of in_region. Equality among regions is defined as expected: two regions are equal whenever they represent the same set of places. Formally,

$$is\_region(r) \wedge is\_region(r') \rightarrow r = r' \equiv \forall p\,\{in\_region^*(p, r) \equiv in\_region^*(p, r')\}$$

Once places have been arranged into regions, we must define connectivity relations among them. We do so by $lifting$ the order relation among places in a path to their corresponding regions. The next section presents how to do so.

8.1.1 Defining paths among regions

Recall that the sort of paths contains the subsorts of topological paths and routes, as stated by axiom 5.5 in chapter 5 (see page 39):

$$\neg\exists pa\,[tpath(pa) \wedge route(pa)]\ \ ,$$

[3]This predicate has been used in the previous chapters. Here we will define the minimum constraints on this predicate.

[4]Directed Acyclic Graph.

where pa is a variable taking values in the sort of paths.

Routes are to topological paths what regions are to topological places. In order to describe abstract relations among paths we use the predicate

$$lifted\_to(pa, pa1)$$

to represent that route $pa1$ is an abstraction of path pa.[5] The predicate $lifted\_to$ plays the same role the predicate in_region does in the context of regions. Consequently we restrict our models to those satisfying the following formulae:

$$\begin{aligned}
&lifted\_to(pa, pa1) \rightarrow route(pa1),\\
&\neg lifted\_to^*(pa, pa),\\
&\{\ min\ lifted\_to^*\ :\\
&\ lifted\_to(pa, pa1) \rightarrow lifted\_to^*(pa, pa1),\\
&\ lifted\_to^*(pa, pa1) \wedge lifted\_to^*(pa1, pa2) \rightarrow lifted\_to^*(pa, pa2),\\
&\}
\end{aligned}$$

The basic rule to lift the order of places in a path to an order among their corresponding regions in a route is defined as follows:

$$\begin{aligned}
&lifted\_to(pa, pa1) \rightarrow\\
&\quad \exists dir \forall p, q\{[order(pa, pos, p, q) \wedge in\_region(p, rp) \wedge\\
&\qquad\qquad in\_region(q, rq) \wedge rp \neq rq\ ] \rightarrow order(pa1, dir, rp, rq)\}
\end{aligned} \tag{8.1}$$

Axiom 8.1 takes into account the fact that different paths can be lifted to a same route. This is accomplished by requiring that the order of places in the positive direction of the path being lifted corresponds to the order of regions in some direction of the route the path is being lifted to. Notice that since $order(pa, pos, p, q) \equiv order(pa, neg, q, p)$ we only need to consider the positive direction of paths in axiom 8.1.

In order for our path lifting method to work, we require a region to be a "path-convex" set of places, that is,

$$order(pa, dir, p, s) \wedge order(pa, dir, s, q) \wedge in\_region(p, r) \wedge in\_region(q, r) \rightarrow in\_region(s, r) \tag{8.2}$$

The next example illustrates how axiom 8.1 works and why we require regions to be "path-convex".

[5]A route can be an abstraction of a set of paths. See example 29, page 116.

Example 28

Consider the path pa depicted in figure 8.1a. Suppose we have the regions, $A = \{a, b\}$, $C = \{c\}$ and $D = \{d, e, f\}$. Let's consider how to lift path pa to path $pa1$. Since place a is before place c in path pa (i.e. $order(pa, dir, a, c)$ is true) then, region A is before region C in path $pa1$ (i.e. $order(pa1, dir, A, C)$ is true).

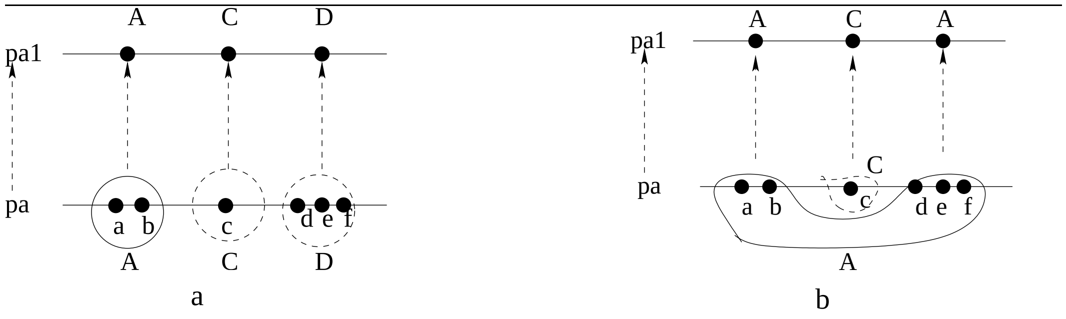


Figure 8.1: (a) Lifting paths. Path pa is lifted to path $pa1$. The order of places in path pa defines the order of their corresponding regions in path $pa1$. (b) Regions have to be "path-convex" in order for our path lifting method to work (see text).

Notice that since a and b belong to the same region A, it is not the case that $order(pa1, dir, A, A)$, although it is the case that $order(pa, dir, a, b)$.

Let's consider a different scenario. Suppose we have two regions, $A = \{a, b, d, e, f\}$ and $C = \{c\}$ (figure 8.1b). Assume we lift path pa to path $pa1$. Since place a is before place c in path pa, region A is before region C in path $pa1$. Similarly, since place c is before place d in path pa, region C is before place A in path $pa1$. Since the order among places in a path is not reflexive, we will have a contradiction.
$\{$*end of example*$\}$

Example 29

Different paths can be lifted to a same route as long as axiom 8.1 is satisfied and the order of regions in the route defines a partial non-reflexive order. For instance, consider the map in figure 8.2, where $RA = \{a, b\}$, $RB = \{c, d\}$, and paths pa and pa' are both lifted to route $pa1$. In addition, assume that $order(pa, pos, a, b)$ and $order(pa', neg, c, d)$ are the case (the positive direction of paths is indicated by arrows in the figure). Then, axiom 8.1 can be satisfied by defining $order(pa1, pos, RA, RB)$. $\{$*end of example*$\}$

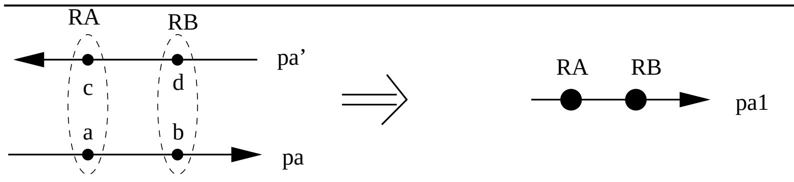


Figure 8.2: A route can be an abstraction of different paths.

The places at which a path enters or leaves a region are used to translate routes from one level of the hierarchy to actual behaviors at the SSH control level (page 118). This in turn implies that we can use hierarchical planning techniques when navigating the environment (see pages 207 and 209). The predicates

$$place\_enter(pa, dir, r, p), \quad path\_leaves(pa, dir, r, p)$$

are used to denote that p is a place at which the path pa enters (leaves) region r, when following pa in direction dir.[6] We explicitly define these predicates as follows:

$$\begin{aligned} place\_enter(pa, dir, r, p) &\overset{def}{\equiv} \\ in\_region(p, r) \wedge \forall q \, \{order(pa, dir, q, p) &\rightarrow \neg in\_region(q, r)\} \end{aligned} \tag{8.3}$$

$$place\_leaves(pa, dir, r, p) \overset{def}{\equiv} place\_enter(pa, -dir, r, p) \tag{8.4}$$

The constraints imposed on the hierarchy of regions and paths do not exclude bizarre structures. Moreover, different hierarchies can be implicitly represented by the predicate in_region. In practice, it is the case that one imposes more structure on the abstraction hierarchy. In particular, "stratified" hierarchies turn out to be the common ones:

Definition 9 (Stratified hierarchies)

The relation in_region defines a *stratified hierarchy* over a set of places SH if there exist a natural number *n*, and a function *level* from SH into $\{0, \ldots, n\}$ such that:

1. $p \in SH \wedge tplace(p) \rightarrow level(p) = 0$.
2. $p \in SH \wedge r \in SH \wedge in\_region(p, r) \rightarrow level(r) = level(p) + 1$.
3. $p \in SH \wedge q \in SH \wedge on(pa, p) \wedge on(pa, q) \rightarrow level(p) = level(q)$.

[6]Given that the order of places is not necessarily a total order, there could exist different places at which a path enters (leaves) a region.

4. $r \in SH \wedge level(r) < n \rightarrow \exists r' \, in\_region(r, r')$.

In addition, we require regions to be connected

$$p \in SH \wedge r \in SH \wedge q \in SH \wedge in\_region(p, r) \wedge in\_region(q, r) \rightarrow connected(r, p, q) \tag{8.5}$$

were the predicate $connected$ is such that

$$\begin{array}{l} \{ \; min \; connected: \\ \;\; in\_region(p, r) \rightarrow connected(r, p, p), \\ \;\; order(pa, dir, p, q) \wedge in\_region(p, r) \wedge in\_region(q, r) \rightarrow connected(r, p, q), \\ \;\; connected(r, p, q) \wedge connected(r, q, s) \rightarrow connected(r, p, s) \\ \} \end{array}$$

{*end of definition*}

Regions in a stratified hierarchy are connected sets of places, such that it should be possible to go from one place to another in a region by only visiting places inside that region.[7] The connectedness property will ensure that a route at any level of a stratified hierarchy has an associated route at level zero of the hierarchy. This property in turns implies that "navigation plans" described at different levels of detail can be translated to actual behavior. The actual behavior is found by repeatedly refining a route at level $k, k > 1$, to a route at level $k - 1$, until finding a route at level 0. Next we define what a refinement for a route is.

Notation 1

A route from region R_0 to region R_n in a stratified hierarchy is denoted by a sequence

$$R_0, (Pa_0, dir_0), R_1, \ldots, (Pa_{n-1}, dir_{n-1}), R_n$$

such that (i) $level(R_i) = level(R_j), \; 0 \leq i, j, \leq n$, and (ii) $order(Pa_i, dir_i, R_i, R_{i+1})$, $0 \leq i < n$.
{*end of notation*}

Definition 10 (Partial refinement)

[7] Some useful region structures are not connected. For instance, the left and right regions of a path are in general not connected.

Given a route $R = \; R_0, (Pa_0, dir_0), R_1, \ldots, (Pa_{n-1}, dir_{n-1}), R_n$, a *partial refinement* of R is a sequence

$$r_0^e, r_0^l, (pa_0, dir_0^*), r_1^e, r_1^l, \ldots, (pa_{n-1}, dir_{n-1}^*), r_n^e, r_n^l$$

such that (i) $in\_region(r_i^e, R_i), \;\; in\_region(r_i^l, R_i)$, $0 \leq i \leq n$, (ii) $lifted\_to(pa_i, Pa_i)$, $0 \leq i < n$, and (iii) $place\_leaves(pa_i, dir_i^*, R_i, r_i^l), \;\; place\_enters(pa_i, dir_i^*, R_{i+1}, r_{i+1}^e)$, $0 \leq i < n$. Notice that the only restrictions on r_0^e and r_n^l are the ones in (i).
{*end of definition*}

A complete refinement for a route R can be found by finding a partial refinement for R, and then finding a route, restricted to R_i, given r_i^e and r_i^l. This last route is guarantee to exist since we require regions to be connected (axiom 8.5).

Definition 11 (Complete refinement)

Given a route $R = \;\; R_0, (Pa_0, dir_0), R_1, \ldots, (Pa_{n-1}, dir_{n-1}), R_n$, p such that $in\_region(p, R_0)$, and q such that $in\_region(q, R_n)$, a *complete refinement of R, given p and q,* is a route $\; r_0, (pa_0, dir_0^*), r_1, \ldots, (pa_{m-1}, dir_{m-1}^*), r_m$ such that for each i, $0 \leq i \leq n$, there exist $i', i'', \; 0 \leq i' \leq i'' \leq m$ satisfying:

1. $r_0 = p$, $r_m = q$.
2. $0' = 0$, $n'' = m$.
3. $r_{0'}, r_{0''}, (pa_{0'}, dir_{0'}^*), r_{1'}, r_{1''}, \ldots, (pa_{(n-1)'}, dir_{(n-1)'}^*), r_{n'}, r_{n''}$ is a partial refinement for R.
4. For all j, $i' \leq j \leq i''$, r_j belongs to R_i (i.e $in\_region(r_j, R_i)$ is the case).

{*end of definition*}

The general algorithm for hierarchical planning with stratified hierarchies is shown in figure 8.3. While our description of refining routes uses a breath-first approach, one could use depth-first too. When using depth-first, one could interleave planning and plan execution so that a complete plan is not necessary in order for the agent to start navigating towards its goal (see page 209). In addition, notice that it is not guaranteed that the algorithm returns an “optimal” path from p to q. Further restrictions on how one creates a refinement and in the hierarchy itself must be imposed for this to be the case. The reader is referred to [Fernandez, 2000] where a complete analysis of hierarchical planning as well as extensions to the ideas here presented can be found.

```
Procedure Hierarchical_planning (p,q)
  ;;; Assume $level(p) = level(q)$,

  find $r$, $p_0, \ldots, p_n$, $q_0, \ldots, q_n$ such that
    (i) $p_0 = p$, $q_0 = q$, $p_n = q_n = r$,
    (i) $p_i \neq q_i$, $0 \leq i < n$,
    (iii) $in\_region(p_i, p_{i+1}),\ \ in\_region(q_i, q_{i+1}),\ 0 \leq i < n.$
  route : = find_route(r,$p_{n-1}$,$q_{n-1}$);
  i = n-2;
  while i $>$ level(p) do
      begin
          route := refine(route,$p_i$,$q_i$);
          i: = i-1;
      end
  return route
{end of procedure}
```

Figure 8.3: **Hierarchical planning algorithm**. $find\_route(r, p, q)$ returns a route from p to q whose places belong to region r. $refine(route, p_i, q_i)$ returns a complete refinement of $route$, given p_i and q_i (see definition above).

8.1.2 One dimensional frames of reference associated with routes

At the SSH metrical level we have to define the distance between regions and the direction between routes. Recall that distance between regions is only explicitly defined for regions in the same route. Direction among routes is derived from the angle between routes at the regions they intersect. Distance and direction could be defined in different ways for regions and routes respectively. For example, for each region a place could be chosen as its representative, and distance among regions is then defined as distance among representatives. In the same vein, it is possible to define "the center of mass" of a region, and then define the distance among regions as the distance among their center of mass.

In the current implementation of the SSH we represent uncertainty on distances by real number closed intervals. The distance uncertainty between two regions is the minimum closed interval containing all distances uncertainties between any two places in the regions.

Formally,[8] [9]

$$\begin{aligned}
& order(pa1, dir, r1, r2) \wedge \\
& a = min\{a' : \exists pa, p, q \;\; [lifted\_to(pa, pa1), in\_region(p, r1), in\_region(q, r2), \\
& \qquad\qquad path\_distance^{\approx}(pa, p, q, [a', b'])]\} \wedge \\
& b = max\{b' : \exists pa, p, q \;\; [lifted\_to(pa, pa1), in\_region(p, r1), in\_region(q, r2), \\
& \qquad\qquad path\_distance^{\approx}(pa, p, q, [a', b'])]\} \\
& \rightarrow path\_distance^{\approx}(pa1, r1, r2, [a, b])
\end{aligned} \tag{8.6}$$

8.1.3 Radial frames of reference associated with regions

In order to define the angle between paths at a region, we associate a two dimensional frame of reference with a region. Each place in the region gets a location with respect to this frame of reference (see section 7.2.3, page 96). We assume that the center of mass of the different place locations is $(0, 0)$.[10] The angle associated with the locations where a path enters or leaves a regions are used to assign the respective path headings with respect to the region:

$$\begin{aligned}
& A = \{ang : \exists pa', pa1', p, q, h_p, h_q \\
& \qquad\qquad [lifted\_to(pa', pa), lifted\_to(pa1', pa1), \\
& \qquad\qquad place\_leaves(pa, dir, r, p), place\_leaves(pa1', dir1, r, q), \\
& \qquad\qquad location2(r, p, l_p), location2(r, q, l_q) \\
& \qquad\qquad directed\_angle(l_p, l_q) = a] \\
& \qquad \} \wedge \\
& ang = min\_int\_cover(A) \\
& \rightarrow angle^{\approx}(r, pa, dir, pa1, dir1, I_{ang})
\end{aligned}$$

where $min\_int\_cover(A)$ denotes the minimum angle interval covering all the angles in the set A. Example 30 (page 122) shows how all the above axioms work in a T like environment.

The resulting hierarchical representation is similar to the Hierarchical Graphs (AH-graphs) representation proposed in [Fernandez and Gonzalez, 1997, Fernandez and Gonzalez, 1998, Fernandez, 2000]. They differ in that ours focuses on topological paths (not just

[8]Notation: Whenever a formula ϕ is a conjunction, $\phi = C_1 \wedge \ldots \wedge C_n$, we will replace $\wedge$ by a comma and write $C_1, \ldots, C_n$.

[9]The predicate $path\_distance^{\approx}(pa, p, q, I_d)$ is used to represent the fact that the closed interval I_d is an estimate of the distance between places *p* and *q* on path *pa*. See page 93.

[10]Once a two dimensional frame of reference is created, the center of mass of the different places can be calculated, and places' locations can be translated so that their center of mass become $(0, 0)$.

edges in a graph), metrical information, and it is defined using an axiomatic theory compatible with the previous description of the SSH. In addition, we have not paid attention to the optimality conditions the hierarchy should satisfy in order to find optimal navigation paths at the bottom level of the hierarchy. Nevertheless, our specifications of regions has been implemented using AH-graphs [Remolina *et al.*, 1999].

8.2 Creating Regions

While we have described how to represent regions in the SSH, the question of how the agent learns such regions is not in the scope of this work. Nevertheless, we consider two general approaches –metrical and topological– that lead to useful hierarchies in office-like environments. In the metrical approach, places that are close to each other define a region. In the topological approach, objects or configurations of objects in the topological map define regions. For instance, in chapter 6 boundary regions were defined by associating with a path its left and right regions. The basic algorithm to create a region hierarchy is presented in figure 8.6 (page 125) . Although the algorithm only uses method (8.7) to create regions, any combination of methods could have been used.

Metrical approach

In this method, a region r is a set of connected places such that if a place p is in r and the distance from p to place q is less than a given threshold, then place q is in r. Formally,

$$in\_region(p,r) \land path\_distance(pa,p,q,d) \land d \leq \epsilon \rightarrow in\_region(q,r) \tag{8.7}$$

The next example illustrates the use of this technique.

Example 30

Consider the environment in figure 8.4, where the agent has identified places a through h. Notice that places b, c, e, f, g are very close one from each other. We can automatically set a threshold such that these places become a region, B. In virtue of axiom 8.7 (and the choice of our threshold) place a gets associated to a region A whose only place is a. Similarly, places d and h get associated with regions $D = \{d\}$ and $H = \{h\}$, respectively. Note that for too large a threshold, or too uniform a distribution of places, all places will fall into the same region, rendering the resulting region structure not useful.

Once these regions have been created, the path $\langle a, b, e, c, d\rangle$ can be lifted to the path $\langle A, B, D\rangle$ and the path $\langle b, f, g, h\rangle$ can be lifted to the path $\langle B, H\rangle$. The resulting topological map has two routes and four regions, as illustrated in figure 8.4b. Notice that the path

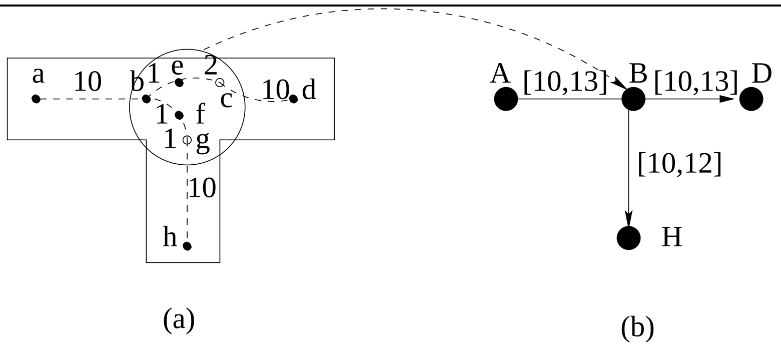


Figure 8.4: Places $\{b, c, e, f, g\}$ are grouped into region B. Distance among places is annotated on the corresponding edges. (b) shows the resulting abstract map associated with (a).

$\langle a, b, e, c, d\rangle$ enters region B at place b and leaves it at place c. As for path $\langle b, f, g, h\rangle$ it enters B at place b and leaves it at place g.

Let's suppose that the heading of path $\langle a, b, e, c, d\rangle$ at place c is 0 degrees. A coordinate frame for region B will indicate then that the heading of path $\langle b, f, g, h\rangle$ at place g is about -90 degrees. Consequently, we can deduce that the angle between paths $\langle A, B, D\rangle$ and $\langle B, H\rangle$ at place B is about -90 degrees. The resulting map is indicated in figure 8.4b. {*end of example*}

Notice that when places are close to each other, it could be the case that the resulting regions contain places that are very far apart from each other. We can handle this case by considering p a constant instead of a variable in axiom 8.7. The resulting method creates a region around a preselected place p. For example, the region associated with Austin will correspond to those towns (places) not farther than 30 miles from Austin.

Topological approach

Topological approaches for creating regions identify subgraphs in the topological map that are "good" candidates for regions. For instance, connected components in the topological map that will be disconnected when removing a path (bridge) from the map are good candidates for regions. Figure 8.5b shows the bridge region map associated with Figure 8.5a. Each office is represented as a place in Figure 8.5b. Next we illustrate how this transformation is done in the case of office A.

Consider the *topological path* connecting places 1 and 6 in figure 8.5a. By removing this path, the subgraph associated with the set of places {1,2,3,4,5} (i.e. office A) becomes

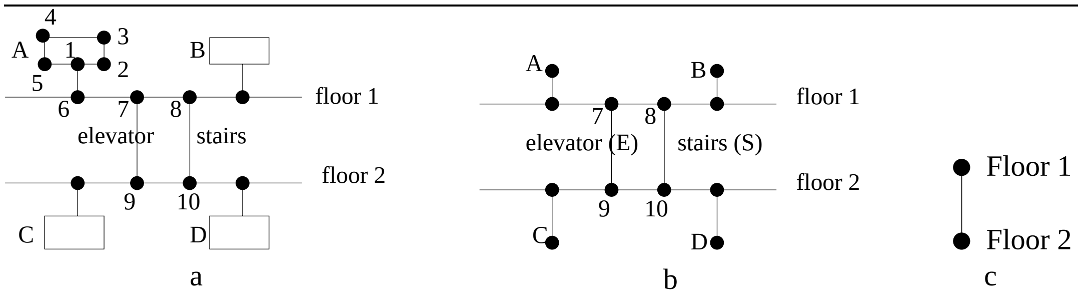


Figure 8.5: (a) Topological map of a two floor building. (b) Bridge region map associated with (a). (c) Bridge region Map with (b).

disconnected from the rest of the map. This fact implies that any route from a place outside A to a place in A necessarily includes place 1. "It makes sense" then to identify all the places in A with place 1 (i.e. create a region associated with A). Notice that we remove topological paths and not edges in the graph. For example, in figure 8.5a we do not consider removing the edge between places 6 and 7. In addition, in order to detect regions we only remove paths of length one. For example, we do not consider removing the topological path including nodes $6, 7$ and 8 (i.e. floor 1).[11]

We still can construct a more abstract map. Notice that removing paths of length one in the map of Figure 8.5b does not produce new disconnected regions. We then consider to remove two paths simultaneously with the restriction that the respective ending of the paths belong to the same other path. For example, consider the path *E* (i.e. the elevator) connecting places 7 and 9, and the path *S* (i.e. the stairs) connecting places 8 and 10. The respective endings of these paths are the set of places {7,8} and {9,10}. From figure 8.5b it is clear that places 7 and 8 belong to a same path (i.e. the corridor in floor 1). Similar conclusion is true for places 9 and 10. By simultaneously removing paths *E* and *S*, the resulting graph has two connected components (i.e. each floor becomes a component). These two components give as a result the map in figure 8.5c.

8.3 Using regions to solve spatial problems

Once a region hierarchy is available, some spatial problems among places can be solved by solving a related problem in terms of the regions associated with these places. For example, in order to decide whether Austin is south of Boston, we might consider the related problem "is Texas south of Massachusetts?". Solving this second problem will be computationally

[11] It is not clear at this point how to remove this constraint.

```
1.  places := set of all places at the topological level.
    paths  := { pa : on(pa,p), p in places}

2.  do until |places| in {0,1}
     {
       regions := emptyset
       newpaths := emptyset

       do until (places = emptyset)   /* create regions */
        {
          pick p in places
          create a new region r. Declare in_region(p,r)
          apply (8.7) until no more places can be grouped in r
          places := places \  {p: in_region(p,r)}
          regions := regions U {r}
          Create a frame of reference for r. Assign locations and headings
              to places and paths in r.
        }

       For pa in paths do    /* lift paths */
        {
          if pa has places in at least two different regions
           then
             {
              create a new path pa'. Declare lifted(pa,pa')
              apply lifting axiom (8.1) to path pa'.
              newpaths := newpaths U {pa'}
             }
        }

       places := regions
       paths := newpaths
     }
```

Figure 8.6: Basic algorithm to create region based topological maps.

less expensive than solving the original one, once the region hierarchy is created.

The key issue when working with a region hierarchy is to establish what properties are preserved when moving up and down the hierarchy. These properties will allow us to establish the soundness of reasoning mechanisms based on regions as well as characterize the typical errors people might make when using regions to infer properties of the places in them. For instance, in the example above, we assumed that both problems, "is Austin south of Boston?" and "is Texas south of Massachusetts?", are equivalent, that is, our region hierarchy preserves orientation information for places in the regions $Texas$ and $Massachusetts$. Notice that this is not necessarily always the case. For example, information of the relative orientation of $Texas$ and $Louisiana$ does not help to decide whether "Austin is south of New Orleans". [12]

8.4 Summary

In this chapter we extended the SSH topological level ontology to include *regions*. Regions are the basic abstraction mechanism the agent has in order to cope with the large number of distinctive states (and so topological places) it will identify in the environment. Connectivity relations among regions were defined according to our lifting axiom 8.1, which states how to abstract the order of places in a path to the order of their corresponding regions in a route (our name for "abstract" paths). We then defined the distance among regions and the angle among routes at their intersection places. In both cases, this metrical information is derived from the metrical information among the places and paths being abstracted to regions and routes, respectively. In order to do so, we require a region to have a frame of reference where places in the region are located.

The constraints imposed in the hierarchy of regions and paths do not exclude bizarre structures. We defined a kind of hierarchies, stratified hierarchies, where it is guaranteed that any route among regions has associated a sequence of actions at the SSH control level. We then presented an algorithm to do so.

Finally, we have defined different criteria for automatically creating regions. While these methods are appropriate for office like environments, we have not addressed the problem of what criteria should the agent use, when should it use them, and how does the agent

[12][Stevens and Coupe, 1978] study illustrates how people incorrectly uses these hierarchies by making conclusions like "San Diego is west of Reno NV" since "California is west of Nevada".

obtains such criteria. Our approach has been completely pragmatic and the goal has been to devise criteria and algorithms for use in robotics.

Chapter 9

Implementation

The goal of this chapter is to illustrate the use of the theoretical ideas presented through this dissertation, as well as to present some problems and techniques not covered by the theory but required for its successful implementation (for instance, how to implement control laws, views, etc.).

Implementing the SSH control level requires one to define a set of control laws and a transformation from sensory input to views. Once this has been done, a set of schemas can be generated by exploring the environment. This set of schemas constitutes the input for the other levels of the SSH. In this chapter we describe the SSH control level implementation, and describe the learned maps in two particular environments: a rectangular environment (Section 9.5.1) and Taylor Hall second floor (Section 9.5.2). The first environment illustrates how boundary information is used to handle view aliasing. The second environment illustrates the kind of map learned in a natural office-like environment. In addition, we define an algorithm to track the different causal and topological maps associated with a set of experiences (if more than one map exists!).

The SSH's implementation described in this chapter has been carried out in Vulcan, our wheelchair robot (Section 9.1). In section 9.2 we describe the control laws used in our experiments. How to define views is described in section 9.3. Section 9.4 presents our implementation of the causal and topological levels. Finally, we present a trace of the exploration of the rectangular and Taylor Hall second floor environments in section 9.5.

9.1 Vulcan

The experiments and algorithms described in this chapter have been implemented in Vulcan, our wheelchair robot (Figure 9.1). The wheelchair base is a Tinman II from the KISS Institute for Practical Robotics.[1] This is a Vector Velocity wheelchair, retrofitted with twelve infrared proximity sensors, seven sonars, two laser rangefinders, and a small embedded computer which manages the drive systems and collects input from the sensors. The principal on-board computer is a dual processor Pentium Pro machine running Debian Linux. Two frame grabber cards allow us to acquire images from dual-monocular or stereo image processing. User interaction (other than joystick commands) is handled through a laptop, also running Linux, which is connected to the main computer via an on board Ethernet network. Two CCD cameras provide our system's visual input, with each camera mounted on a directed perceptions pan-tilt head.

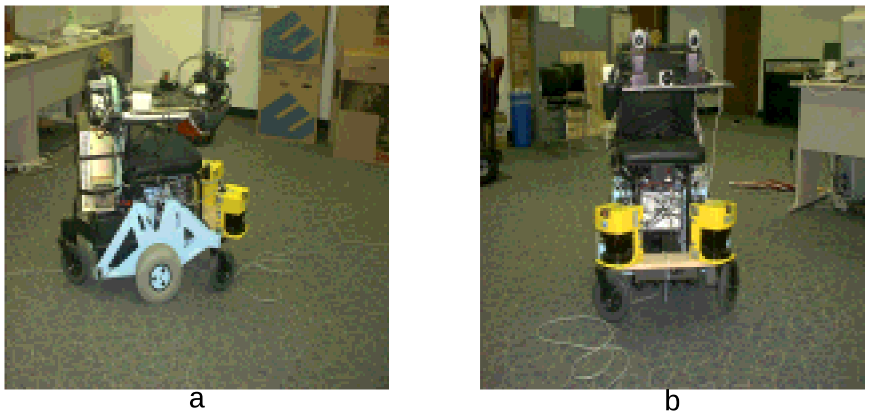


Figure 9.1: (a) Vulcan, our wheelchair robot. (b) Laser rangefinders configuration.

In the experiments, only the laser rangefinders have been used for sensory input. A laser reading provides information about the actual distance measured by a laser rangefinder. We assume that the robot has an egocentric frame of reference w.r.t. which the location of objects is determined. Hereafter, we assume this system of coordinates is such that the positive x axis extends forward and the positive y axis extends to the left. Thus, positive angles are to the left, negative angles are to the right, and angle zero is straight ahead.

[1]http://www.kipr.org .

9.2 Control laws

In this section we describe our boundary following and hill climbing control laws. A boundary following control law keeps the robot at a given distance from an object while moving along the object's boundary. In our examples, the objects correspond to walls in the environment, though the robot does not know of the existence of such objects. Hill climbing control laws localize the robot within its current local neighborhood (see section 9.2.3, page 134).

Our boundary following control law specifies the value of the controlled variable w (the robot's rotational speed) according to the rule

$$\omega = \frac{1}{v}\left[-k_\theta v\theta - k_e e\right] \tag{9.1}$$

where the observed variables e, θ and v are defined as follows: e is the difference between the distance to the object and the desired distance to keep from the object; θ is the orientation w.r.t. the boundary; v is the forward velocity (figure 9.2). The constraint

$$k_e = \frac{k_\theta^2}{4}$$

guarantees that the system is critically damped.

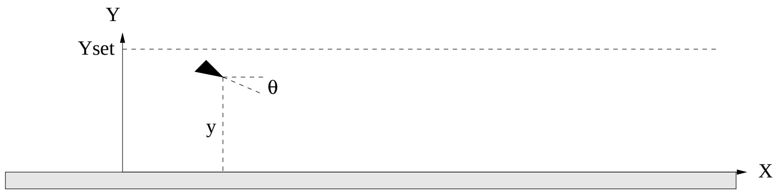


Figure 9.2: The robot is at position (x, y) and orientation θ. $e = y - y_d$, where y_d is the desired distance from the wall.

Rule 9.1 is used to obtain different behaviors: follow a wall, follow a corridor, go through a doorway, hill climb to a corridor intersection, etc.. In all these cases, a different *observer* is used to calculate the values of e and θ needed by 9.1.[2] The function of the observer is twofold: first, it is in charge of filtering noisy data from the sensors, match an object model (if exists), and calculate the values of e and θ. Second, the observer determines the applicability associated with the control law. Figure 9.3 illustrates the general observer architecture we use in our implementation. In the next subsections we describe the wall following, corridor following, go through doorway and hill climb to intersection behaviors.

[2]In our implementation the value of v is kept constant.

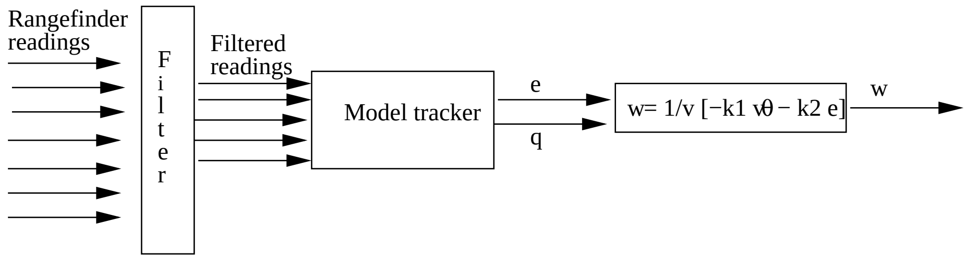


Figure 9.3: Observer architecture.

9.2.1 Following a wall/corridor

In order to calculate the values of e and θ line segment representing the wall(s) are identified in the rangefinder data. The parameters defining these line segments are used to calculate e and θ (see figure 9.9, page 139).[3] In section 9.3.1 we present our algorithm to extract line segments from rangefinder data. In order to use this algorithm, one specifies the laser rangefinder angular sector defining the set of scans that will be considered when extracting a segment. For example, a left wall will be associated with the angular sector $(45^o, 135^o)$ while a right wall is associated with $(-135^o, -45^o)$. Figure 9.4 illustrates how a corridor is perceived by the robot's laser scanners.

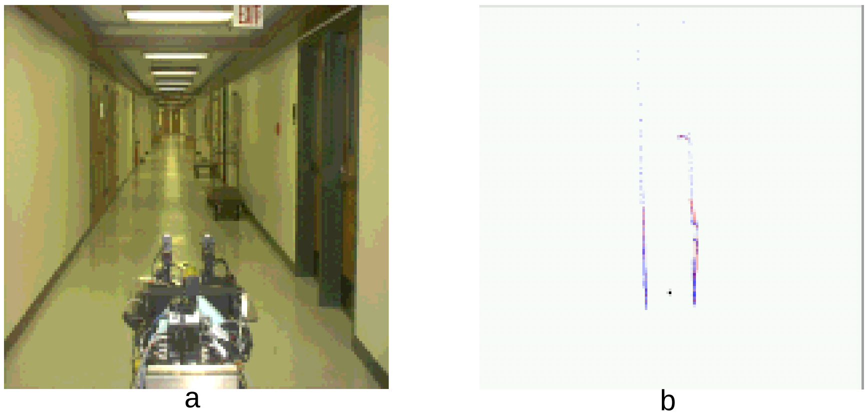


Figure 9.4: (a) A typical corridor scenario. (b) Two dimensional plot of the laser scan associated with **(a)** and its corresponding segments.

In order to initially identify the position of the wall(s) to follow, the robot selects

[3]Let ρ and θ' be the parameters defining a line segment as in figure 9.9. Then, when following a boundary on the robot's right, $e = \rho - y_d$ and $\theta = -\theta'$, where y_d is the desired distance from the boundary.

the largest segment on the right (and left) whose angle is about zero and whose distance from the robot location is less than 2(meters). After that, a *tracker* is assigned to each wall so that the agent can easily decided when the walls disappear.

Different implementations of the tracker are possible. The simplest one corresponds to storing the parameters of the line segment representing the wall being tracked, and selecting as the current wall model the "closest" line segment to the stored parameters. In order to make this implementation more robust, the tracker does not calculate e and θ directly from the segment parameters, but uses these parameters and other information (e.g. previous values of these parameters, a model of the boundary) to calculate an estimation for e and θ. This estimation can use a median or mean filter in both e and θ[4] [Gonzalez and Woods, 1992] or a *Kalman filter* [Gershenfeld, 1999] associated with rule 9.1.[5] In the last case, this Kalman filter is derived as follows: for small values of θ we have that $\dot{e} = v\theta$, which allow us to write the dynamical system associated with this controller as

$$\begin{bmatrix} \dot{\theta} \\ \dot{e} \end{bmatrix} = A \begin{bmatrix} \theta \\ e \end{bmatrix} \tag{9.2}$$

$$A = \begin{bmatrix} -k_\theta v & -\frac{k_e}{v} \\ v & 0 \end{bmatrix} \tag{9.3}$$

We can obtain a discrete system as

$$x_{t+\delta_t} = (I + A\delta_t)x_t \tag{9.4}$$

where $x_t = \begin{bmatrix} \theta_t \\ e_t \end{bmatrix}$. Moreover, we assume that there is some noise in the process, w_t, with constant distribution process G. Thus, the dynamical model of our Kalman filter is defined by

$$x_{t+\delta_t} = (I + A\delta_t)x_t + Gw_t \tag{9.5}$$

As for the measurement model of the Kalman filter, we assume that e and θ can be observed (since they are derived directly from the line segment parameters), and consequently the reading at time t, z_t, obeys the equation

$$z_t = Ix_t + v_t \tag{9.6}$$

where v_t is the measurement noise, which we assume has constant covariance R.

[4]In our experiments we use a window size of three values, so we store the last two values of (say) e, add a new one, and output the median (mean) of these three values.

[5]This filter step is different from the one shown in figure 9.3. There, a filter is applied to the raw laser data, while here, a filter is applied to the line segment parameters.

9.2.2 Going through doorways

A doorway can be described by a given configuration of line segments, for instance, two collinear segments with "open space" between them, or as two perpendicular segments with "open space" between them. The condition of two segments being collinear or perpendicular can be checked from the corresponding segment parameters. To check that "open space" exists between two segments, one has to choose the end points of the segments that define the doorway, and then to check for "large" rangefinder readings between these two points. As an example of a doorway described by two perpendicular segments with "open space" between them, consider a robot facing a door as in figure 9.5. As an example of a doorway described by two collinear segments with "open space" between them, consider a robot facing a door as in figure 9.6.

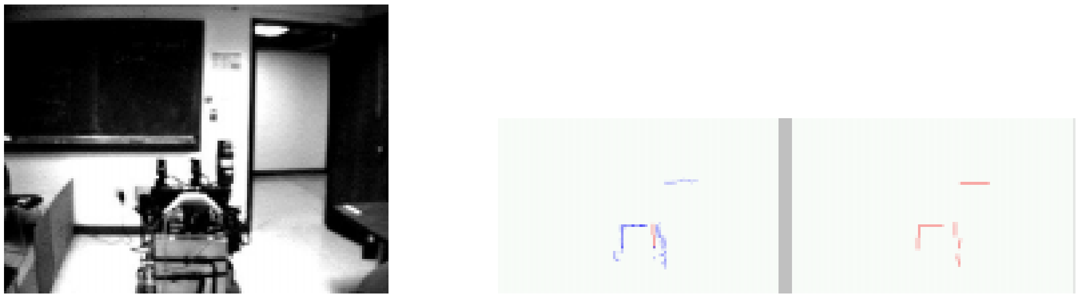


Figure 9.5: The robot is facing a door from inside a room toward a corridor. On the right a two dimensional plot of the laser scan with an extra line indicating the bisector of the doorway recognized by the robot. The rightmost figure shows the segments used to detect the doorway. A doorway is recognized by finding two perpendicular segments with open space between them.

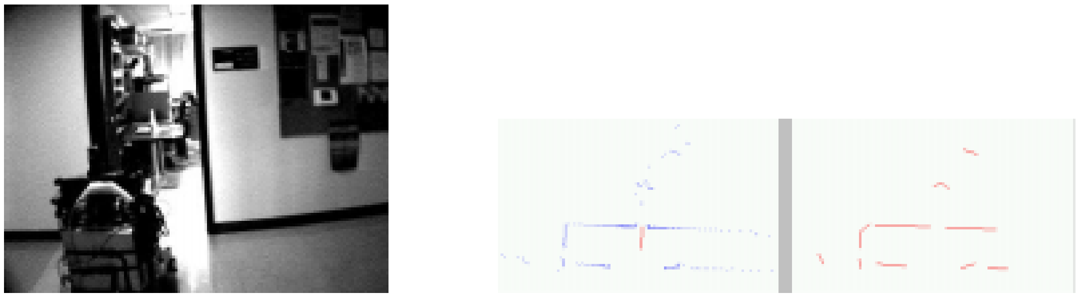


Figure 9.6: The robot is looking inside a room from a corridor. On the right a two dimensional plot of the laser scan with an extra line indicating the bisector of the doorway recognized by the robot. The rightmost figure shows the segments used to detect the doorway. A doorway is recognized by finding two collinear segments with open space between them.

In order to go through a doorway, the bisector of the doorway is used as the boundary to follow. This idea works not only for doorways but for going through open space in general: for instance, for the robot to get into a corridor (figure 9.7), the two corners defining the corridor (which are the two closest minima in the rangefinder readings) can be used as landmarks such that the bisector of the line connecting them defines the line to follow.

9.2.3 Hill Climbing

Hill climbing strategies localize the robot with respect to its local neighborhood. After hill climbing the robot will be at the distinctive state associated with the environment state where the hill climbing was started. Theoretically, this distinctive state corresponds to the hill climbing control law's fixed point. However, in practice it is not always possible to reach the fixed point exactly. An approximation to this fixed point is appropriated as long as one guarantee that the robot recognizes the same view after hill climbing to the same distinctive state (see example 31, page 140).

For the purpose of this dissertation we implemented a general hill climbing control law that place the robot equidistant from three or more objects. The main issue when implementing this control law is that the robot does not know in advance the location of these objects, nor do these objects necessarily appear in its current sensory horizon (the robot cannot "see" all the closest objects at once). Consequently, the robot has to create a local frame of reference where it localizes (by using odometry) and sensory input is integrated over time. Next we illustrate the general method in the case where the robot hill climbs to a corridor intersection after detecting the end of a corridor (see Figure 9.7).

1. A frame of reference is created.

2. The robot orients itself facing in the direction perpendicular to the line segment linking the contact points of the two closest objects. Let θ_0 denote the resulting orientation (w.r.t. to the local frame of reference).

3. Sensor readings are integrated into the current frame of reference and the distances to the closest objects are calculated. If three or more objects are equidistant, then the robot stops and the algorithm goes to the next step. Otherwise, the robot follows the perpendicular bisector of the segment linking the contact points of the two closest objects. The control law used for this purpose is as in equation 9.1. The values of e and θ are calculated from the location of the robot as well as the location of the contact points of the two closest objects. This location is the one in the local frame of reference. The robot's location in the frame of reference is updated using odometry.

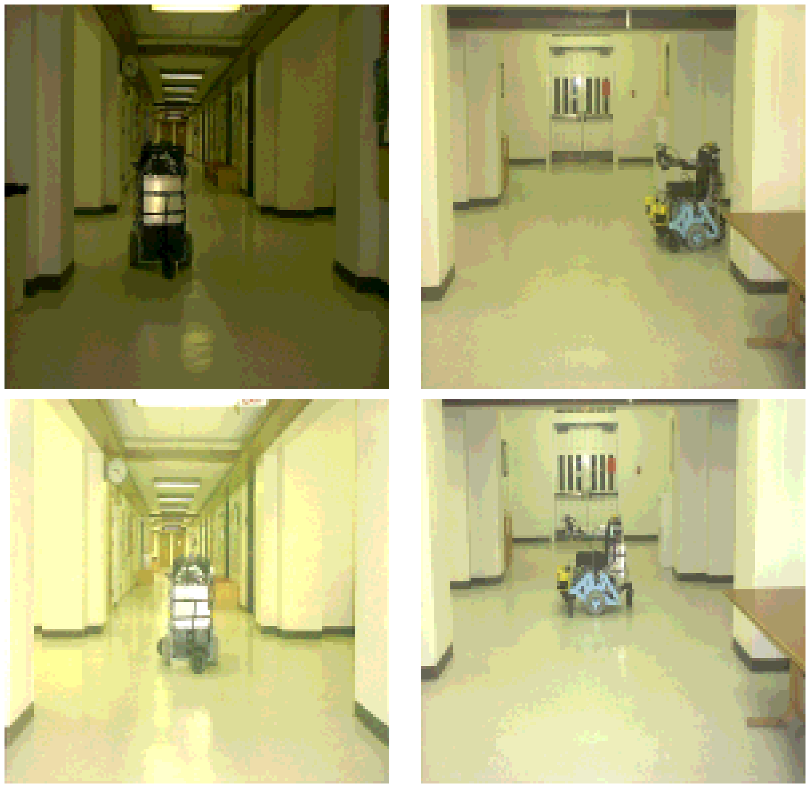


Figure 9.7: Hill climbing to a corridor intersection. The top row illustrates two views of the initial robot location before hill climbing. The bottom row illustrates the corresponding views after hill climbing.

The robot keeps moving until a third object is detected or until it has traveled more than a given distance threshold.

4. Once the robot stops, the contact points of the closest equidistant objects are ordered according to their orientation with respect to the local frame of reference. The angles (orientation) of these contact points as long as the bisector of the angles between consecutive contact points are considered, and the robot reorients by facing the angle closest to θ_0.

The last step of the algorithm is necessary so that the robot keeps as much as possible the same orientation as when it started the hill climbing. In office-like environments this step maximizes the chances that a travel action can be executed from the resulting distinctive state, and consequently paths with more than two places (i.e. corridors) can be identified.[6] For instance, when hill climbing in a T intersection as described in figure 9.8, the robot will change its orientation while localizing the third closest object (corner *B* in the figure). If the robot does not reorient, a distinctive state will be created with the robot facing directly into corner *B*. No travel action can be executed from this distinctive state and consequently no corridor will be identified.

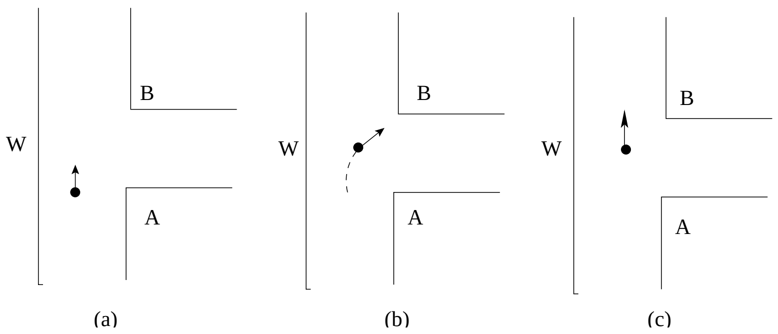


Figure 9.8: Hill climbing in a T intersection. The robot starts having as reference object corner *A* and wall *W*. While keeping equidistant from these two objects, it detects corner *B*. The robot proceeds until it is equidistant from objects *W*, *A* and *B*. Finally, the robot reorients facing the bisector of the angle determined by objects *W* and *B*, which is the closest orientation to the one when starting the hill climbing.

In the algorithm implementation, an object corresponds to a line segment extracted from the local map created while doing the hill climbing. The local map is implemented as an occupancy grid [Elfes, 1987, Thrun, 1998]. The robot location as well as the sensory integration is done according to the standard occupancy grid equations [Thrun, 1998]. In order to extract line segments (features) from the local map, a Hough transform method was used [Vandorpe *et al.*, 1996, Anousaki and Kyriakopoulos, 1999]. This method is widely used in image processing in order to extract the edges from a camera image [Gonzalez and Woods, 1992].

The general hill climbing strategy overcomes some of the limitations of the hill climbing implementation proposed by Lee [Lee, 1996]. In Lee's work, an automata describes local environment transitions as sensory information (sonar readings) become available to the robot. The final states of such automata correspond to the "type" of local environments the robot can recognize. The method here proposed does not assumes particular types of local environments, but rather assumes the existence of more powerful sensors (laser rangenfinders), which combined with standard techniques to represent local environ-

[6]This last reorient step was missing in Lee's SSH implementation [Lee, 1996].

ments allow the robot to keep track of objects not visible in the current sensory image.

9.3 Views

Each distinctive state has associated a view symbol representing what the distinctive state "looks like". In order to create such view the agent examines the sensory input associated with the distinctive state neighborhood. Views are created by considering the set of segments associated with the rangefinder scan at the agent environment location.[7] Next we describe our implementation and provide some experimental results (see example 31, page 140) showing the appropriateness of the method.[8]

Each laser scan has associated a sequence of line segments $(ls_1, \ldots, ls_n)$ which constitutes its view representation. This sequence of segments is found by extracting the set of line segments associated with the scan (see Section 9.3.1), and ordering them according to their angle.

The problem of finding how similar two scans are is reduced to the one of finding how similar two finite sequences of line segments are. An *embedding* from $(ls1_1, \ldots, ls1_n)$ into $(ls2_1, \ldots, ls2_m)$ is an *increasing* function from a set in $choose(min(n,m), n)$[9] into $\{1, \ldots, m\}$. The goodness of an embedding e is defined as follows:

$$Goodness(e) = \frac{1}{n} \sum_{i=1}^{min(n,m)} sim(ls1_i, ls2_{e(i)}) \tag{9.7}$$

where $sim(l1, l2)$ is a positive function returning a value in $[0,1]$ indicating how similar line segments $l1$ and $l2$ are. We will provide a definition for sim later.

The similarity of $s1 = (ls1_1, \ldots, ls1_n)$ w.r.t. $s2 = (ls2_1, \ldots, ls2_m)$ is given by

$$Sim(s1, s2) = \max_e Goodness(e) \quad ,$$

the value of the best embedding of $s1$ into $s2$. Finally, the match between $s1$ and $s2$ is given by

$$match(s1, s2) = min(Sim(s1, s2), Sim(s2, s1)) \quad .$$

In order to calculate $match(s1, s2)$ an adaptation of the *longest common subsequences* algorithm can be done [Cormen *et al.*, 1993]. Next we define how to compare line

[7]Different methods can be considered to create such view: create an occupancy grid of the dstate neighborhood, train a neural net, etc.. See [Duckett and Nehmzow, 2000] for a comparative study of these different methods.

[8]The work in this section is joint effort with Micheal Hewett.

[9]Choose(k,n) denotes the subsets of $\{1, \ldots, n\}$ that have k elements.

segments.

Each line segment is described by four parameters (ρ, θ, l, mp) where θ is the angle in $[-\pi, \pi]$ of the line normal measured with respect to the *x* axis, ρ is the signed distance of the line to the origin (see figure 9.9), l denotes the length of the segment, and mp denotes the midpoint coordinates of the segment.

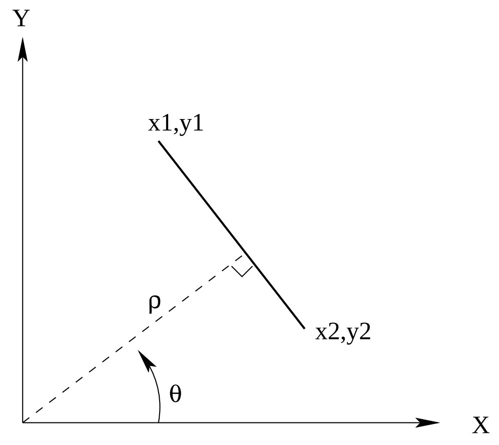


Figure 9.9: ρ-θ representation of a line.

The similarity between two line segments $l1 = (\rho_1, \theta_1, l_1, mp_1)$ and $l2 = (\rho_2, \theta_2, l_2, mp_2)$, $sim(l1, l2)$, is defined as follows:

$$sim(l1, l2) = mean(f_a(a_1, a_2)) \ , \quad a \in \{\rho, \theta, l, mp\} \tag{9.8}$$

where the different functions f_a are normal distributions functions, each with their own mean and standard deviation:

$$f_a(x, y) = N(\mu_a, \sigma_a)(x - y) \ \ .$$

In general it is the case that the robot is off by an angle every time it hill climbs to the same distinctive state. Consequently, two configurations of segments associated with a distinctive state neighborhood will have to be rotated before the view matching described above. We take into account this rotation by considering the function

$$match^*(s1, s2) = min(Sim^*(s1, s2), Sim^*(s2, s1)) \ \ ,$$

where

$$Sim^*(s1, s2) = max_{e,\theta} Goodness^*(e, \theta) \ \ ,$$

$$Goodness^*(e,\theta) = \frac{1}{n} \sum_{i=1}^{min(n,m)} sim(ls1_i^{\theta}, ls2_{e(i)}) \ \ ,$$

and l^{θ} denotes the line segment resulting of rotating l by an angle θ. The next example illustrates the appropriateness of the view matching defined in this section.

Example 31

Figure 9.10 illustrates different segment configurations associated with the distinctive state resulting of hill climbing to a corridor intersection (see figure 9.7, page 135). Table 9.11 shows the similarity values for these different configurations.

The values in table 9.11 represent the intraclass similarity corresponding to the view (class) associated with the distinctive state. In this table, scan 3 is taken at the ideal distinctive state location: the wheelchair perfectly aligns with the corridors (see figure 9.7, page 135). Notice how the similarity values decrease as the orientation changes with respect to scan 3. For instance, scans 1 and 5 differ by about 10° and their similarity value is only 0.58.

While not shown, we have found that for most of our experiment 0.6 is an appropriate threshold for deciding whether two segment configurations correspond to the same view (class). As illustrated in section 9.5.2 (page 176), this threshold is way above the similarity values obtained for scans taken at different distinctive states (i.e. extra-class similarities). While it is an interesting problem to formulate how the agent learns this threshold, or for that matter, how it decides how many classes (views) should be associated with its sensory input at distinctive states, that learning task is out the scope of this work.
$\{$*end of example*$\}$

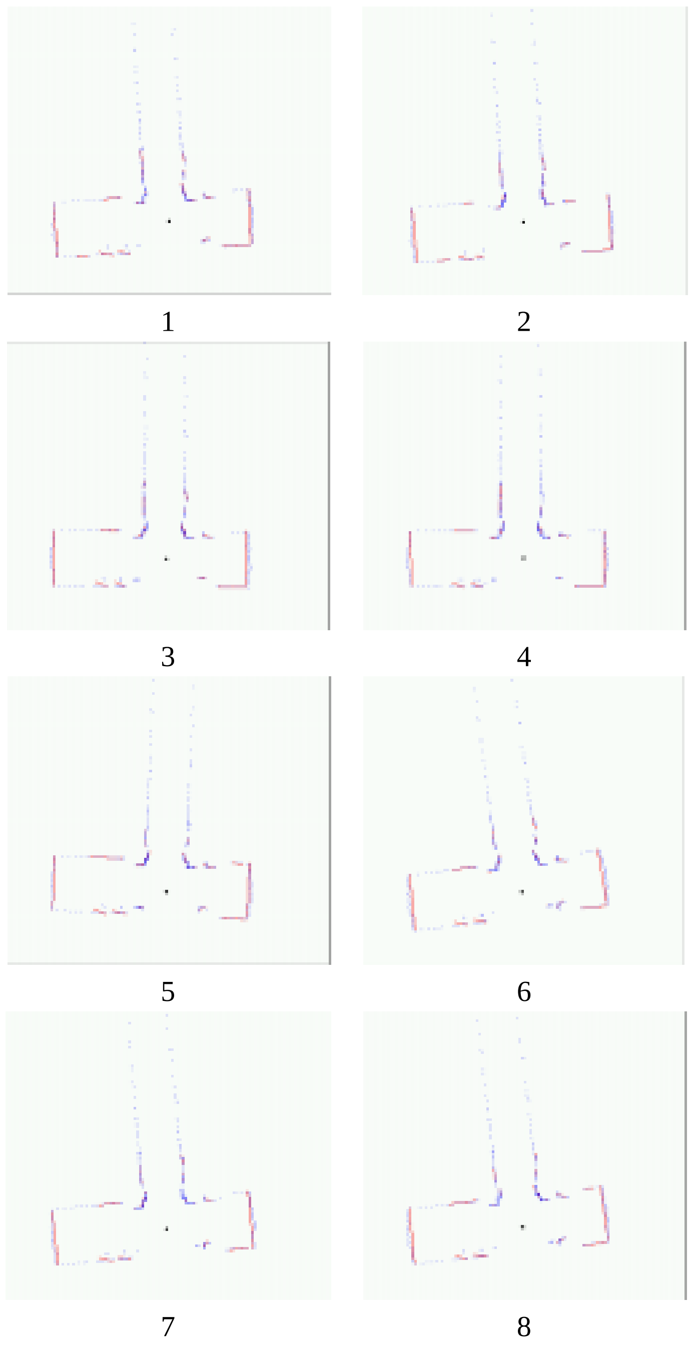


Figure 9.10: Different laser range-finder images at the same distinctive state neighborhood associated with a corridor intersection (figure 9.7, page 135).

Scan	1	2	3	4	5	6	7	8	mean
1	1.0	0.86	0.76	0.78	0.58	0.75	0.71	0.71	0.65
2	0.86	1.0	0.77	0.77	0.56	0.73	0.78	0.67	0.66
3	0.76	0.77	1.0	0.87	0.81	0.71	0.62	0.63	0.69
4	0.78	0.77	0.87	1.0	0.77	0.65	0.57	0.60	0.64
5	0.58	0.56	0.81	0.77	1.0	0.61	0.59	0.69	0.66
6	0.75	0.73	0.71	0.65	0.61	1.0	0.80	0.86	0.64
7	0.71	0.78	0.62	0.57	0.59	0.80	1.0	0.76	0.64
8	0.71	0.67	0.63	0.60	0.69	0.86	0.76	1.0	0.63

Figure 9.11: Similarity values for each pair of configurations in figure 9.10.

9.3.1 Line segment extraction

In this section we describe our algorithm to extract line segments from a laser rangefinder scan. Let *r* denote a laser reading. We use the following notation to refer to the information associated with *r*: *range(r)* denotes the value (distance) returned by the laser rangefinder, *angle(r)* denotes the angle in the robot's egocentric frame of reference associated with the rangefinder location, and *point(r)* denotes the (x, y) coordinates in the robot's egocentric frame of reference associated with the vector defined by *range(r)* and *angle(r)*. A *laser scan* is an ordered sequence of laser readings. The readings in a laser scan *s* are ordered such that the $angle(s(i)) < angle(s(i+1))$. That is to say, readings in a scan are ordered right to left w.r.t. the robot's frame of reference.[10] A *line segment ls* is represented by its pair of extreme points $(p1(ls), p2(ls))$.

Our algorithm to find the set of segments associated with a laser scan is as follows:

1. Identify qualitative clusters in the scan ranges.
2. Associate a segment with each identified cluster.
3. Merge consecutive collinear segments.
4. Remove segments whose length is less than a given threshold $k\_min\_segment\_length$.
5. Merge consecutive collinear segments.

Qualitative clusters in a laser scan are identified by finding discontinuities, local maxima and local minima in the scan ranges. Noise in the laser data is taken in account

[10]The readings in a scan could have been ordered left to right. The important assumption is that they are ordered.

when identifying these clusters. Discontinuities are detected by finding laser readings that are close in the angle dimension but whose ranges are farther apart than a given threshold $k\_discontinuity\_jump$.

Local maxima and minima are detected by considering the range of a laser reading as a function of the angular sector's length associated with the scan. In order to identify monotonic (i.e. increasing, decreasing, or constant) regions in the data we adapted the qualitative filtering algorithm proposed in [Kay *et al.*, 1999, Rinner and Kuipers, 1999]. This algorithm uses a window of k_window_size consecutive readings to determined the different trends in the data. Whether a trend is declared to be increasing, decreasing or constant, is determined by comparing the slope of points in a window against a threshold k_std_dev.

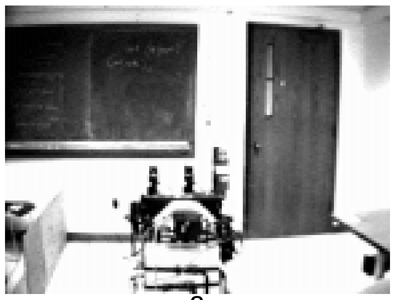
For each cluster in the laser scan, a segment is created by finding the least square fitting of the two dimensional set of points associated with the readings in the cluster. Clusters are ordered by the angle associated with their corresponding laser readings. The segments associated with each cluster preserve this order. This order is important since at the merging step of the algorithm we merge consecutive collinear segments, rather than arbitrary collinear segments.

Consecutive close collinear segments are merged to form new segments. Collinear segments whose distance is less than a given threshold $k\_merge\_distance$ are merged. A tolerance in the angle ($k\_collinear\_angle\_threshold$) and distance from the origin($k\_collinear\_range\_threshold$) are used to decide whether two segments are collinear.

Once segments have been created, we remove those whose length is less than $k\_min\_segment\_length$. It is possible that by removing a segment two different segments become consecutive collinear segments. To care for this case, we apply the merging step one more time.

Example 32

Suppose the robot is facing a wall straight ahead as indicated in figure 9.12a. When plotting the laser readings in a two dimensional plane we can observe three segments defining the local environment of the robot (see figure 9.12b).



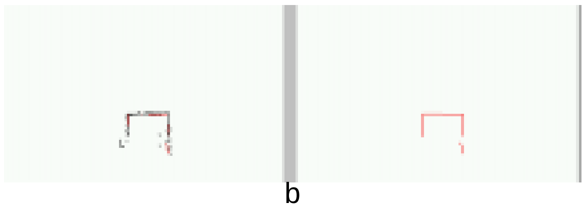
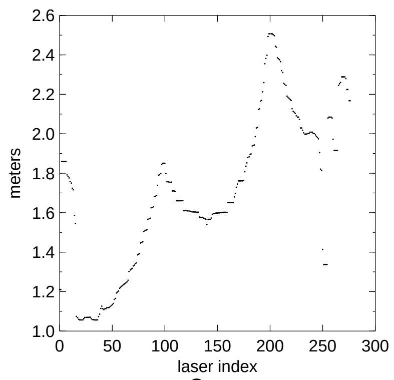


Figure 9.12: (a) The robot is facing a wall straight ahead. There is an artificial wall on the left and a book shelve on the right (not seen in the picture). (b) On the left a two dimensional plot of the laser rangefinder scan associated with the environment in a. On the right, the corresponding four segments extracted from the laser scan.

Figure 9.13a shows the result of plotting left to right the laser readings. The corresponding trends associated with this data are shown in figure 9.13b. In the laser range domain the left wall is sensed by a decreasing trend in the range (laser indexes 10 to 25), reaching a minimum at the closest point on the robot's left (laser indexes 26 to 28) , follow by an increasing trend in the range which ends at the corner between the left and front wall (laser indexes 28 to 100).



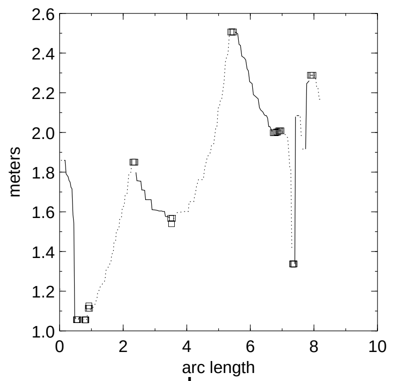


Figure 9.13: (a) Laser readings associated with the robot location in 9.12. (b) Clusters associated with the laser readings.

Once the different trends in the ranges are detected, their boundaries define clusters to which line segment are associated. These line segments are usually part of a final segment to be extracted. The algorithm's merging step puts these different parts together.

For example, the left wall in figure 9.12 has associated three clusters indicated by the laser indexes 10, 25, 28 and 100. The resulting segments associated with these clusters are all collinear, and consequently merged into one segment.
{*end of example*}

Example 33

In order to identify qualitative clusters in the scan ranges, first we detect discontinuities (jumps) in the data. These discontinuities define clusters of readings on which *decreasing*, *increasing* or *constant* trends are detected. These trends are the only patterns in the data used to define segments. While example 32 may suggest that it is possible to associate a segment with more complex patterns in the data (e.g. "U" patterns), this is not necessarily the case. For instance consider the case in which the robot has partial access to different rooms (figure 9.14a). Figure 9.14b shows the laser rangefinder data and the segments found by the algorithm.

As shown in figure 9.15a, the laser ranges have discontinuities and some walls are not perceive in a "U" pattern as in example 32. However, the different up, down, and constant trends in the data do define part of the segments to be extracted. Whether two trends are part of a same line segment is decided by assigning line segments to each trend and then checking for these segments to be collinear.
{*end of example*}

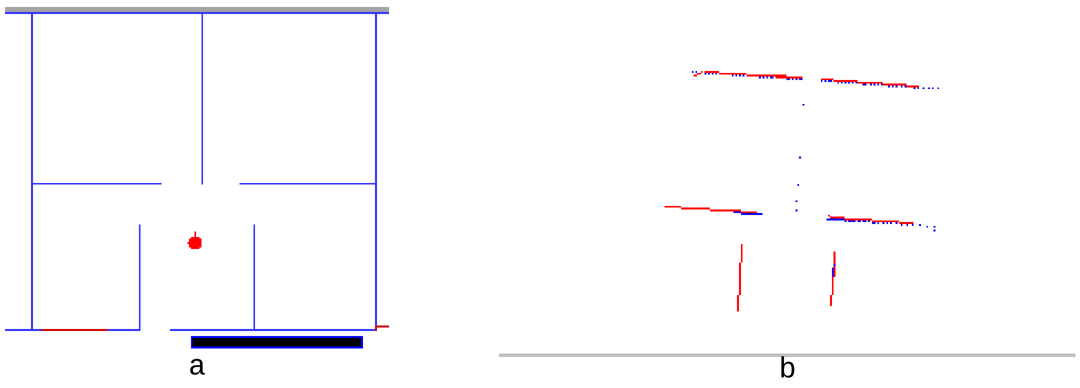


Figure 9.14: (a) A robot has partial access to different rooms. (b) The extracted segments.

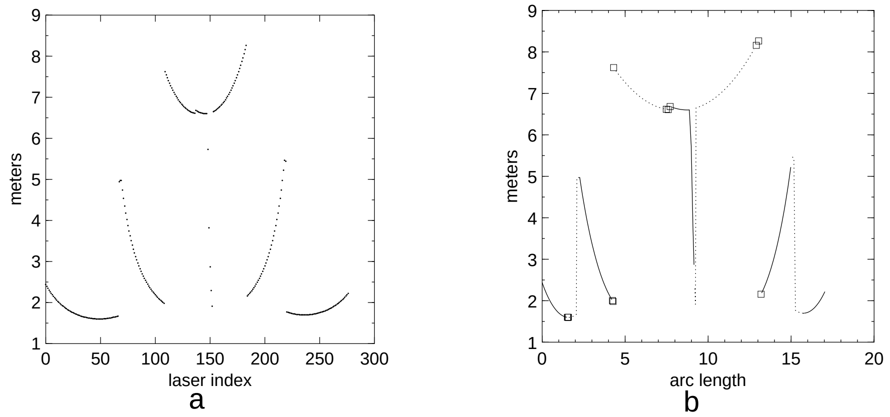


Figure 9.15: (a) Laser readings associated with the robot location in 9.14a. (b) Clusters associated with the laser readings.

9.4 Causal/Topological/Metrical levels

In this section we present an implementation for calculating the SSH causal and topological maps (i.e. models of the theories $CT(E)$ and $TT(E)$ respectively). The SSH metrical level algorithms were presented in chapter 7.[11] A logic program implementing the circumscriptive theory defining the SSH causal level (Chapter 4) is presented in section 9.4.1. It is possible to calculate the models of $TT(E)$ by a logic program similar to the one used for $CT(E)$. However, the number of grounding rules associated with such a program turns out to be prohibited for practical applications. Fortunately, the problem of calculating the models of $TT(E)$ can be stated as a *"best first"* search. In section 9.4.2 (page 152) we present this algorithm.

9.4.1 Using Logic Programming to implement the SSH causal level

Given a set of experiences E, the models of the theory $CT(E)$ (Chapter 4, page 26) indicate under what circumstances it is possible to consider two distinctive states as referring to the same environment state. In order to calculate these models, we define a logic program whose answer sets [Gelfond and Lifschitz, 1991] are in a one to one correspondence with the models of $CT(E)$.[12] Recall that the theory $CT(E)$ is defined as follows:

$$\begin{array}{l}
COMPLETION(E)\ , \\
Axioms\ 4.1 - 4.22\ , \\
\langle ds, a, ds' \rangle \wedge \langle ds, a, ds'' \rangle \rightarrow ds' = ds'' \\
CEQ\_block\ = \\
\{\ max\ \ ceq : \\
\quad ceq(ds, ds), \\
\quad ceq(ds_1, ds_2) \rightarrow ceq(ds_2, ds_1), \\
\quad ceq(ds_1, ds_2) \wedge ceq(ds_2, ds_3) \rightarrow ceq(ds_1, ds_3), \\
\quad ceq(ds_1, ds_2) \rightarrow ViewAt(ds_1, v) \equiv ViewAt(ds_2, v), \\
\quad ceq(ds_1, ds_2) \wedge \langle ds_1, a, ds_1' \rangle \wedge \langle ds_2, a, ds_2' \rangle \rightarrow ceq(ds_1', ds_2') \\
\}
\end{array}$$

The logic program Π we will consider is defined as follows:

$$\begin{array}{rcl}
p(X,Y,X,Y) & \leftarrow & . \\
p(X,Y,X2,Y1) & \leftarrow & p(X,Y,X1,Y1), ceq(X1,X2).
\end{array}$$

[11]The use of metrical information while building the topological map is illustrated in section 9.5.1, page 169.

[12]The logic program is implemented in Smodels [Niemelä and Simons, 1997]. The output produced by Smodels is parsed in order to keep track of the different SSH causal models associated with a set of schemas. See an illustrative trace in example 34.

$$
\begin{array}{rcl}
p(X,Y,X1,Y2) & \leftarrow & p(X,Y,X1,Y1), ceq(Y1,Y2). \\
p(X,Y,X2,Y2) & \leftarrow & p(X,Y,X1,Y1), cs(X1,A,X2), cs(Y1,A,Y2). \\
p(X,Y,Y1,X1) & \leftarrow & p(X,Y,X1,Y1). \\
p(X,Y,X1,Y2) & \leftarrow & p(X,Y,X1,Y1), p(X,Y,Y1,Y2).
\end{array}
$$

$$
\begin{array}{rcl}
dist(X,Y) & \leftarrow & p(X,Y,X1,Y1), viewAt(X1,V), not\ viewAt(Y1,V). \\
dist(X,Y) & \leftarrow & p(X,Y,X1,Y1), not\ viewAt(X1,V), viewAt(Y1,V).
\end{array}
$$

$$\leftarrow \quad not\ ceq(X,X). \tag{9.9}$$

$$\leftarrow \quad ceq(X,Y), not\ ceq(Y,X). \tag{9.10}$$

$$\leftarrow \quad ceq(X,Y), ceq(Y,Z), not\ ceq(X,Z). \tag{9.11}$$

$$\leftarrow \quad ceq(X,Y), viewAt(X,V), not\ viewAt(Y,V). \tag{9.12}$$

$$\leftarrow \quad ceq(X,Y), not\ viewAt(X,V), viewAt(Y,V). \tag{9.13}$$

$$\leftarrow \quad not\ ceq(X1,Y1), ceq(X,Y), cs(X,A,X1), cs(Y,A,Y1). \tag{9.14}$$

$$\leftarrow \quad not\ ceq(X,Y), not\ dist(X,Y). \tag{9.15}$$

where the variables X and Y range over distinctive states and the variable V ranges over views. Rules 9.9-9.11 require *ceq* to be an equivalence class. Rules 9.12-9.14 are the counterpart of the axioms inside CEQ_block. In order to define the maximality condition of ceq, the auxiliary predicate $p(X,Y,X1,Y1)$ is introduced. This predicate reads as *"If X and Y were the same, then $X1$ and $Y1$ would be the same"*. The predicate $dist(X,Y)$ defines when distinctive states X and Y are distinguishable. Constraint 9.15 establishes the maximality condition on *ceq*: $ceq(X,Y)$ should be the case unless X and Y are distinguishable. In appendix F (page 236) we prove that the answer sets of this logic program represents the different SSH causal models.

Example 34

Consider the environment depicted in Figure 9.16. The agent visits the different distinctive states as suggested by their numbers in the figure. Assume that the different corners have the same views (i.e. view(1) = view(4)=view(8), view(3) = view (7) = view (11)). Below we show a trace of our algorithm illustrating how we keep track of all possible worlds consistent with the agent's experiences. Recall that *this algorithm uses only causal information*. The building of the topological map will be described in section 9.4.2 (page 152).

For the purpose of the example, we generate a new distinctive state every time the agent experiences a view (for instance, distinctive states 10 and 2 both occur at the same physical environment state).

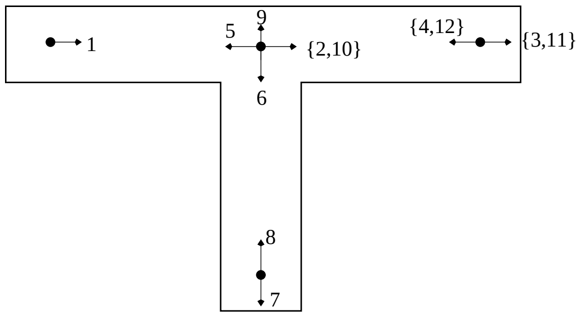


Figure 9.16: T-environment used to illustrate how the different SSH causal models evolve as the agent navigates the environment. The numbers shown are the dstates, not the views.

```
break[9]=>(set! tour3 '(v1 ml v2 ml v3 turn-a v1 ml v4 turn-l v5
ml v3 turn-a v1 ml v6 turn-r v2 ml v3 turn-a v1))

break[10]=>(process-tour tour3)

Adding new Schema <SCHEMA-1:: (DS-1,v1), ml, (DS-2,v2)>
Checking for Inconsistent worlds
 - [world:: ]  << Causally consistent >>

Current Worlds

1 [world:: ]
```

For each world we only show the *ceq* equivalence classes that contain more than one distinctive state. In particular, `[world:: ]` **denotes the world where all distinctive states are different.**

```
Adding new Schema <SCHEMA-2:: (DS-2,v2), ml, (DS-3,v3)>
Checking for Inconsistent worlds
 - [world:: ]  << Causally consistent >>

Current Worlds
```

```
1 [world:: ]

Adding new Schema <SCHEMA-3:: (DS-3,v3), turn-a, (DS-4,v1)>
Checking for Inconsistent worlds
 - [world:: ]  << Causally consistent >>

Current Worlds

1. [world:: (DS-4 DS-1)]
```

At this point the agent cannot distinguish `DS-1` `from` `DS-4`, **and so the agent has only one model of the world where these two distinctive states are the same.**

```
Adding new Schema <SCHEMA-4:: (DS-4,v1), ml, (DS-5,v4)>
Checking for Inconsistent worlds
 - [world:: (DS-4 DS-1)]  <<  Causally Inconsistent >>
 >> Not Eq relations learned >>>  {not-eq:{DS-1:: (DS-4)} {DS-4:: (DS-1)}}

Current Worlds

1 [world:: ]
```

Once the agent travels from `DS-4` to `DS-5` and observes view $V4$, distinctive states `DS-1` and `DS-4` become distinguishable, since executing ml starting at `DS-1` takes the agent to a distinctive state with view $V2$ which is different from $V4$. In addition, the agent learns that in any possible world, `DS-1`neq`DS-4`.

```
Adding new Schema <SCHEMA-5:: (DS-5,v4), turn-l, (DS-6,v5)>
Checking for Inconsistent worlds
 - [world:: ]  << Causally consistent >>

Current Worlds

1 [world:: ]

Adding new Schema <SCHEMA-6:: (DS-6,v5), ml, (DS-7,v3)>
Checking for Inconsistent worlds
 - [world:: ]  << Causally consistent >>

Current Worlds
```

```
1 [world:: (DS-7 DS-3)]

Adding new Schema <SCHEMA-7:: (DS-7,v3), turn-a, (DS-8,v1)>
Checking for Inconsistent worlds
 - [world:: (DS-7 DS-3)]  << Causally consistent >>

Current Worlds

1 [world:: (DS-8 DS-4)(DS-7 DS-3)]
2 [world:: (DS-8 DS-1)]
```

Notice that multiple possible worlds are created once the agent reaches `DS-8`. In the model in which `DS-7`=verb+DS-3+, verb+DS-8+ has to be *ceq* to verb+DS-4+ since actions are deterministic.

```
Adding new Schema <SCHEMA-8:: (DS-8,v1), ml, (DS-9,v6)>
Checking for Inconsistent worlds
 - [world:: (DS-8 DS-4)(DS-7 DS-3)]  <<  Causally Inconsistent >>
 >> Not Eq relations learned >>>  {not-eq:{DS-4:: (DS-8)} {DS-8:: (DS-4)}}
 - [world:: (DS-8 DS-1)]  <<  Causally Inconsistent >>
 >> Not Eq relations learned >>>  {not-eq:{DS-1:: (DS-8)} {DS-8:: (DS-1)}}

Current Worlds

1 [world:: ]
```

Once the agent travels to `DS-9` only one world is possible, the one in which all distinctive states are indeed different.!!

```
Adding new Schema <SCHEMA-9:: (DS-9,v6), turn-r, (DS-10,v2)>
Checking for Inconsistent worlds
 - [world:: ]  << Causally consistent >>

Current Worlds

1 [world:: (DS-10 DS-2)]

Adding new Schema <SCHEMA-10:: (DS-10,v2), ml, (DS-11,v3)>
Checking for Inconsistent worlds
 - [world:: (DS-10 DS-2)]  << Causally consistent >>
```

```
Current Worlds

1 [world:: (DS-11 DS-3)(DS-10 DS-2)]
2 [world:: (DS-11 DS-7)]

Adding new Schema <SCHEMA-11:: (DS-11,v3), turn-a, (DS-12,v1)>
Checking for Inconsistent worlds
 - [world:: (DS-11 DS-3)(DS-10 DS-2)]  << Causally consistent >>
 - [world:: (DS-11 DS-7)]  << Causally consistent >>

Propagate-not-eq

Current Worlds

1 [world:: (DS-12 DS-8)(DS-11 DS-7)]
2 [world:: (DS-12 DS-1)]
3 [world:: (DS-12 DS-4)(DS-11 DS-3)(DS-10 DS-2)]
```

All the models above are possible since at the causal level turn and travel actions do not convey any spatial meaning. Should we consider topological information, only model 3 above will be possible (see example 15, page 53).
{*end of example*}

9.4.2 Calculating the models of TT(E)

The algorithm for calculating the models of $TT(E)$ can be stated as a "*best first*" search.[13] The states of the search correspond to partial models of $TT(E)$.[14] At each step of the search a schema $\langle\, ds, a, ds'\,\rangle$ has to be explained. Either the identity of ds' can be proved or a search branch is created for every previously known distinctive state ds'_i that cannot be proven to be different from ds'.[15] In the branch where $ds'_i \overset{teq}{=} ds'$ is the case, $ds'_j \overset{teq}{\neq} ds'$, $i \neq j$ are also asserted.[16] An additional brach is created where $ds' \overset{teq}{\neq} ds'_j$ are asserted. This branch represents the possibility that ds' is indeed different from previously

[13]The models of $TT(E)$ correspond to the topological maps associated with a set of experiences E. See chapter 5, page 36.

[14]A partial model of $TT(E)$ is a model of $TT(E')$, for some $E' \subseteq E$.

[15]We assume that at each state of the search, the identity of the schema's context (i.e. ds in $\langle\, ds, a, ds'\,\rangle$) is known.

[16]The notation $ds_1 \overset{teq}{=} ds_2$ states that ds_1 and ds_2 are "equal" according to the equivalence relation teq. Recall that teq plays the role of equality in the theory $TT(E)$.

known dstates. The next state to explore is the one that is minimal according to the order associated with the circumscription policy for $TT(E)$. This search algorithm is described in figures 9.17 and 9.18.[17]

```
Find-Models (Schemas S)
{
  ;; S = $s_0, \ldots, s_n$ ; sequence of schemas such that
  ;;     $result(s_i) = context(s_{i+1})$
  ;;
  ;; pmodels-to-explore = ordered queue of partial models to explore.
  ;; models = list of total models for S.
  pmodels-to-explore = $\emptyset$ ;
  models = $\emptyset$ ;
  pmodel = create-new-pmodel(S);
  insert(pmodel,pmodels-to-explore) ;
  while pmodels-to-explore $\neq \emptyset$ do
    begin
      pmodel = get-next-pmodel(pmodels-to-explore);
      s = get-next-schema(pmodel);
      Explain(pmodel,s) ;
      if (inconsistent(pmodel) $\vee$ has-extensions(pmodel)) then skip;
      else if total-model(pmodel) then insert(pmodel, models);
      else insert(pmodel,pmodels-to-explore);
    end
  return models;
}
```

Figure 9.17: **Best first search algorithm used to calculate the models of TT(E)**. The ordered queue *pmodels-to-explore* contains *consistent* partial models (pmodels) to be expanded. At each step of the search, a minimal partial model is picked and the next schema from its list of associated schemas is explained. A pmodel has extensions when a branch has been created while explaining a schema. A pmodel is a *total-model* when it has no more schemas to explain. Figure 9.18 defines how a pmodel explains a schema and when extensions are created.

[17]A search state is implemented by a partial model, *pmodel*. Branches in the search are represented by creating *extensions* for the current search state (pmodel). That $pmodel'$ is an extension of $pmodel$ implies that $pmodel'$ inherits from $pmodel$ all known objects and facts. Partial models are described in page 155.

```
Explain (pmodel, s)
{ ;; s is a schema $\langle\, ds, a, ds' \,\rangle$
  candidates = {};
  branches = false;
  if $\neg$ known-result(pmodel,s)
  then begin
     candidates = possible-equal-dstates(pmodel,s);
     if candidates $\neq$ {}
       then branches = create-possible-extensions(pmodel,s,candidates)
    end
  if $\neg$ branches then Assert-schema (pmodel,s);
}
Known-result(pmodel, s)
{ ;; s is a schema $\langle\, ds, a, ds' \,\rangle$
  ;; The notation $obj \in pmodel$ indicates that object $obj$ is
  ;; known in the partial model $pmodel$.
  return $ds' \in pmodel \;\vee \exists\; ds^* \in pmodel\; \left[\langle\, ds^*, a, ds'^* \,\rangle \in pmodel \wedge ds^* \overset{teq}{=} ds\right]$ ;
}
Assert-schema (pmodel, s)
{ ;; s is a schema $\langle\, ds, a, ds' \,\rangle$. $ds$ is known in $pmodel$
  assert $s \in pmodel$;
  if $\neg$ known-result(pmodel,s)
   then begin
     assert $ds' \in pmodel$;
     Create places and paths needed to explain $s$.
   end
   else begin
     pick $ds'^*$ s.t. $\exists ds^* \in pmodel\; \left[ds^* \overset{teq}{=} ds \wedge \langle\, ds^*, a, ds'^* \,\rangle \in pmodel\right]$ ;
     assert $ds' \overset{pmodel}{=} ds'^*$ ;
   end
}
```

Figure 9.18: **Explaining a schema**. *known-result(pmodel,s* $= \langle\, ds, a, ds' \,\rangle$*)* is the case when the equality class for ds' can be deduced in the partial model $pmodel$. *Possible-equal-dstates(cntx,s)* returns dstates known in $pmodel$, having the same view as ds' and that cannot be proven different from ds' in $pmodel$. For each $ds'' \in candidates$, *create-possible-extensions(pmodel,s,candidates)* creates an extension of $pmodel$ where $ds' \overset{teq}{=} ds''$ is the case. If some of these extensions are consistent, then *create-possible-extensions* also creates an extension where ds' is different from the dstates in *candidates*. In this last case the function returns $false$ otherwise it returns $true$. If not extension is created, then s is asserted in $pmodel$. This accounts to declare ds' to be known in $pmodel$ and create the places and paths that explain s according to the axioms of the topological theory $TT(E)$.

The three key steps in the search are (figure 9.18): creating a set of possible candidates to branch (*possible-equal-dstates*), generating a set of extensions when needed (*create-possible-extensions*), and explaining a schema in a given partial model (*assert-schema*). Another important issue is to detect when a partial model becomes inconsistent. We use the predicate $inconsistent(pmodel)$ to denote this fact and the rule[18]

$$x \overset{pmodel}{=} y \wedge x \overset{pmodel}{\neq} y \rightarrow inconsistent(pmodel) \; .$$

In the next sections we will show how to rewrite the axioms in the topological theory so they can be fed to a theorem prover to deduce equality and inequality relations.[19] In section 9.4.3 (page 161) we present an illustrative trace of the algorithm.

Partial Models

In addition to a list of schemas to explain, a partial model has associated a set of SSH objects (i.e. distinctive states, schemas, places, paths) that are known in the model. We use the relation $known\_objects(pmodel, obj)$ to indicate that obj belongs to the known objects of $pmodel$. We also denote this relation by $obj \in pmodel$. From a formal point of view, the objects known in a partial model define the set of constant symbols used in the topological language associated with such pmodel.

That two objects, obj_1 and obj_2, are equal or different in a pmodel is represented by the relations $equal\_objects(obj_1, obj_2, pmodel)$ and $not\_equal\_objects(obj_1, obj_2, pmodel)$ respectively. We also use the convenient notations $obj_1 \overset{pmodel}{=} obj_2$ and $obj_1 \overset{pmodel}{\neq} obj_2$. The equality relation among distinctive states corresponds to when two distinctive states are topologically equal. When the model is understood, we just use the notation $obj_1 = obj_2$ instead of $obj_1 \overset{cntx}{=} obj_2$.

The basic relation among pmodels is the one of *extensions*. That $pmodel'$ is an extension of $pmodel$ implies that all known objects and facts in $pmodel$ are known objects and facts in $pmodel'$.[20]

In the implementation, all the topological predicates are *relativized* with respect to a pmodel. In order to do so, all the predicates in the topological theory have a last extra argument for a pmdodel. For instance, instead of writing $at(ds, p)$ we write $at(ds, p, pmodel)$.[21]

[18]The notation $x \overset{pmodel}{=} y$ is an abreviation for $(x \in pmodel) \;\wedge\; (y \in pmodel) \;\wedge\; x = y$.
[19]Our "theorem prover" in this case is Algernon [Crawford and Kuipers, 1991].
[20]$pmodel'$ inherits from $pmodel$ all known objects and facts. See page 157.
[21]When the pmodel is understood we drop it from the predicate arguments.

$at(ds, p, pmodel)$ is the case when $at(ds, p)$ is true in the partial model $pmodel$ (i.e. $pmodel \models at(ds, p)$).

Our logic for partial models takes the basic ideas developed in the area of formally reasoning about contexts [McCarthy and Buvač, 1998]. We do not provide a formal account of contexts, but rather use the notations and conventions mentioned above.

Create candidates

Possible-equal-dstates(pmodel,s=$\langle\, ds, a, ds'\,\rangle$*)* returns a list of dstates that are possible equal ($\overset{teq}{=}$) to ds'. These are dstates known in $pmodel$, having the same view as ds' and that cannot be proven different from ds' in $pmodel$. We use the predicate $no\_possible(s, ds'', pmodel)$ to indicate that ds' cannot be equal to ds'' in the partial model $pmodel$.[22] In order to filter out ds'' we use the following rules:[23]

$$s = \langle\, ds, turn, ds'\,\rangle \wedge at(ds, p) \wedge at(ds'', q) \wedge p \neq q \rightarrow no\_possible(s, ds'') \tag{9.16}$$

$$\begin{aligned}&[s = \langle\, ds, travel, ds'\,\rangle \wedge along(ds, pa, dir) \wedge along(ds'', pa1, dir1) \wedge \\ &\qquad \neg\,[pa = pa1 \wedge dir = dir1]] \rightarrow no\_possible(s, ds'')\\ &[s = \langle\, ds, travel, ds'\,\rangle \wedge along(ds, pa, dir) \wedge at(ds, p) \wedge at(ds'', q) \wedge \\ &\qquad order(pa, dir, q, p)] \rightarrow no\_possible(s, ds'')\end{aligned}$$

$$\begin{aligned}&[s = \langle\, ds, (turn\ \theta), ds'\,\rangle \wedge at(ds, p) \wedge at(ds'', p) \wedge radial(p, ds, h) \wedge \\ &\qquad radial(p, ds'', h1) \wedge h1 \neq (h + \theta \pmod{360^\circ})] \rightarrow no\_possible(s, ds'')\end{aligned} \tag{9.17}$$

The rules above are derived from the axioms in our theory. For instance, rule 9.16 is derived from the fact that each distinctive state is at a unique place, and distinctive states that are related by turn actions are at the same place. As for rule 9.17, distinctive states have a unique heading at the radial frame of reference associated with a place. We have simplified the presentation of the rule and assume that the headings of dstates at places can be calculated by adding angles $\bmod 360^\circ$. In a robust implementation of these rules, uncertainty associated with metrical information should be taken into account.

In order to use the rules above, other rules are needed to determine that two SSH objects (i.e. dstates, places, paths) are different. These rules include:

$$view(ds_1, v_1) \wedge view(ds_2, v_2) \wedge v_1 \neq v_2 \rightarrow ds_1 \neq ds_2 \tag{9.18}$$

[22]Recall we assume that ds' is not known in $pmodel$. When generating candidates, in order to rule out ds'' we prove the stronger assumption

$$\forall x \in pmodel\ \left[s = \langle\, ds, a, x\,\rangle \rightarrow x \neq ds'' \right] \quad .$$

$no\_possible(s, ds'')$ is understood to represent this last formula.

[23]Recall that rules are *relativized* to a particular partial model.

$$\langle\, ds, turn, ds' \,\rangle \rightarrow ds \neq ds' \tag{9.19}$$

$$order(pa, dir, p, q) \rightarrow p \neq q \tag{9.20}$$

$$radial(p, ds1, h1) \wedge radial(p, ds2, h2) \wedge h1 \neq h2 \rightarrow ds1 \neq ds2 \tag{9.21}$$

$$position1(pa, dir, p1, pos1) \wedge position1(pa, dir, p2, pos2) \wedge pos1 \neq pos2 \rightarrow p1 \neq p2 \tag{9.22}$$

$$LeftOf(pa, dir, p) \wedge on(pa, q) \rightarrow p \neq q \tag{9.23}$$

$$LeftOf(pa, dir, p) \wedge on(pa1, p) \rightarrow pa \neq pa1 \tag{9.24}$$

Rules 9.18 and 9.19 rely on our basic assumption that dstates have a unique view and turn actions link different distinctive states. Rule 9.20 uses the fact that paths are not circular in order to conclude that if p is before q then p and q must be different. Rules 9.21 and 9.22 use radial and one dimensional frames of reference to conclude inequality of dstates and places, respectively. Rules 9.23 and 9.24 use boundary relations in order to distinguish places and paths respectively.[24]

Finally, rules to prove equality among SSH objects include:

$$at(ds, p) \wedge at(ds, q) \rightarrow p = q \tag{9.25}$$

$$along(ds, pa, dir) \wedge along(ds, pa1, dir1) \rightarrow pa = pa1 \wedge dir = dir1 \tag{9.26}$$

The rules to derive equality and inequality relations are not necessarily *complete* in the following sense. It is possible that $no\_possible(s, ds'')$ cannot be proved, and so ds'' will be a valid candidate. A new extension of $pmodel$ will be created in which $ds' \overset{teq}{=} ds''$ is asserted. However, when asserting s in that extension, it can then be proved that $ds' \overset{teq}{\neq} ds''$. We further discuss this issue in the next section.

Creating extensions

For each $ds'_i \in candidates$, *create-possible-extensions(pmodel,s=*$\langle\, ds, a, ds' \,\rangle$*,candidates)* creates a possible extension for $pmodel$ where $ds' \overset{teq}{=} ds'_i$ is the case. If some of these extensions are consistent, then *create-possible-extensions* also creates an extension where ds' is known and different from the dstates in *candidates*. In this last case the function returns

[24]In Algernon [Crawford and Kuipers, 1991] some of these rules are implemented as *forward-chaining (if-added)* rules (e.g. rules 9.18 and 9.19) and others as *backward-chaining (if-needed) rules* (e.g. rules 9.23 and 9.24). The problem with if-added rules is that they could derive a large number of useless truths. For instance, if rule 9.23 were implemented as an if-added rule, Algernon's engine will prove that places p and q are different for any place q on a path pa and place p to the left of pa. While this is the case, in general we want to use boundary relations to derive inequalities among particular pairs of places rather than indiscriminately between all pair of places. Algernon should not (try to) prove at front inequalities among all places in the map!. We provide no further discussion on this issue which applies in general to rule base systems that use forward chaining rules.

$false$ otherwise it returns $true$.

Given $ds'_i \in candidates$, a new extension $pmodel'_i$ is created where $ds' \overset{pmodel'_i}{=} ds'_i$ is the case. In addition, $ds' \overset{pmodel'_i}{\neq} ds'_j$ is also asserted for $ds'_j \in candidates, ds'_j \neq ds'_i$. The $extension(pmodel, pmodel'_i)$ relation among partial models is then asserted. That $pmodel'_i$ is an extension of $pmodel$ implies that all known objects and facts in $pmodel$ are known objects and facts in $pmodel'_i$. This inheritance property of extensions can be implemented in Algernon by rules like the next ones:

$$\begin{aligned}
&extension(pmodel, pmodel1) \wedge known\_objects(pmodel, obj) \\
&\qquad\qquad\qquad\qquad \to known\_objects(pmodel1, ojb) \\
&extension(pmodel, pmodel1) \wedge equal\_objects(pmodel, obj1, obj2) \\
&\qquad\qquad\qquad\qquad \to equal\_objects(pmodel1, obj1, obj2) \\
&extension(pmodel, pmodel1) \wedge not\_equal\_objects(pmodel, obj1, obj2) \\
&\qquad\qquad\qquad\qquad \to not\_equal\_objects(pmodel1, obj1, obj2) \\
&at(ds, place, pmodel) \wedge extension(pmodel, pmodel1) \to at(ds, place, pmodel1) \\
&\ldots
\end{aligned}$$

In addition to state that $extension(pmodel, pmodel'_i)$ is the case, s becomes the next schema that has to be explained by $pmodel'_i$. It is possible that by explaining s at this point (by asserting s in $pmodel'_i$), $pmodel'_i$ becomes inconsistent and so it should not be considered further in the search.[25] Should $pmodel'_i$ become inconsistent, then one has to delete the $extension(pmodel, pmodel'_i)$ relation. We call **look ahead** to this extra step of explaining s in $pmodel'_i$ (see figure 9.19).

Assert schema

Assert-schema(pmodel, s) creates the places and paths needed to explain s. An example of the rules used to explain s is presented in figure 9.20.

Instead of asserting $s = \langle\, ds, a, ds' \,\rangle$ in $pmodel$, the algorithm asserts $s^* = \langle\, ds^*, a, ds'^* \,\rangle$ where ds^* and ds'^* are the $\overset{teq}{=}$ representatives for the equality classes of ds and ds' in $pmodel$. Asserting a schema in Algernon corresponds to creating the frame (object) representing the schema. Forward and backward chaining rules derived from the SSH topological theory are then evaluated, and places and paths needed to explain s are created (see figure 9.20).

[25]This could be the case since rules to filter out candidates may not be complete. See previous section.

```
(defun ssh-assert-extension (pmodel pmodel1 equality
                              &key (next-experience nil) (look-ahead *ssh-look-ahead*))
   (tell `((:the ?pm (name ?pm ,pmodel) (isa ?pm ssh-pmodels))
           (:the ?pm1 (name ?pm1 ,pmodel1) (isa ?pm1 ssh-pmodels))
           (extension ?pm ?pm1)
           (extension-of ?pm1 ?pm)
     ))

   ;;; The above statement will fire Algernon rules to inherit
   ;;; objects and facts from pmodel to pmodel1.

   (ssh-assume-equals pmodel1 (car equality) (cdr equality))
   (if next-experience (ssh-set-next-pmodel-experience pmodel1 next-experience))
   ;;; the look ahead step
   (when (and look-ahead next-experience)
     (ssh-inherit-from-sub-model pmodel1)
     (ssh-assert-experience pmodel1 next-experience)
     (cond
     ((ssh-is-inconsistent pmodel1)
         (tell `((:the ?pm (name ?pm ,pmodel) (isa ?pm ssh-pmodels))
                 (:the ?pm1 (name ?pm1 ,pmodel1) (isa ?pm1 ssh-pmodels))
                 (:delete (extension ?pm ?pm1))
                 (:delete (extension-of ?pm1 ?pm))))
         (ssh-remove-pmodel pmodel1))
     (t (ssh-get-next-pmodel-experience pmodel1))
         ;; remove next-experience from the list of schemas
         ;; to be explained by pmodel1.
     )
   )
)
```

Figure 9.19: **Assert-extension**. Given an equality $(ds'.ds'_i)$, $pmodel1$ is an extension of $pmodel$ where $ds' \overset{pmodel1}{=} ds'_i$ is the case. When look ahead is used, the next schema associated with $pmodel$ is asserted in the extension $pmodel1$. Should this render $pmodel1$ inconsistent, the $extension$ relation is deleted from the database. The function *ssh-remove-pmodel* updates the search queue by removing $pmodel1$ from it.

```
(defun ssh-assert-experience (pmodel e)
    (let ( schema ds-context ds-result act)
        (setf schema
            (first (ask ‘((:the ?e (name ?e ,e) (isa ?e experiences))
                            (E-context-ds ?e ?ds) (E-result-ds ?e ?ds1) (E-action ?e ?a))
                        :collect ’(?ds ?a ?ds1))))
        (setf ds-context (ssh-find-or-create-equal pmodel (first schema)))
        (setf ds-result (ssh-find-or-create-equal pmodel (third schema)))

        ;;; ssh-find-or-create-equal will ensure that ?ds and ?ds1 are known in pmodel

        (setf act (second schema))
        (create-schema :pmodel pmodel
            :initial-ds ds-context :result-ds ds-result
            :action act)
)

(defun Create-Schema (&key (pmodel nil)(action nil)
                          (initial-ds nil) (result-ds nil))
    ;; create frame representing schema ⟨initial-ds, action, result-ds⟩.
)
```

;;; Example of Algernon rules used to explain schemas.

```
(tell ’((:rules Schemas
        ((S-action ?s ?a) (A-type ?a turn) (S-context-ds ?s ?dsc ?cntx)
         (S-result-ds ?s ?dsr ?pmodel)
         (at ?dsc ?p ?pmodel)              ;;; create place if necessary
         →                                ;;; turning keeps the agent in the same place
         (at ?dsr ?p ?pmodel)
        ))
    ’((:rules distinctive-states
        ((at ?ds ?p ?pmodel)
         ←               ;;; a distinctive state is always at a place
         ;;; check that ?ds is not at a place already
         (:fail (:boundp ?p)) (:boundp ?pmodel) (:fail (:retrieve (at ?ds ?a-p ?pmodel)))
         ;;; if so then create place
         (:bind ?p (Create-place (ssh-create-place-name) :pmodel ’?pmodel))
        ))
)
```

Figure 9.20: Asserting a schema in Algernon corresponds to creating the frame (object) representing the schema. Forward and backward chaining rules derived from the SSH topological axioms will then be evaluated, and places and paths needed to explain s will be created (if needed).

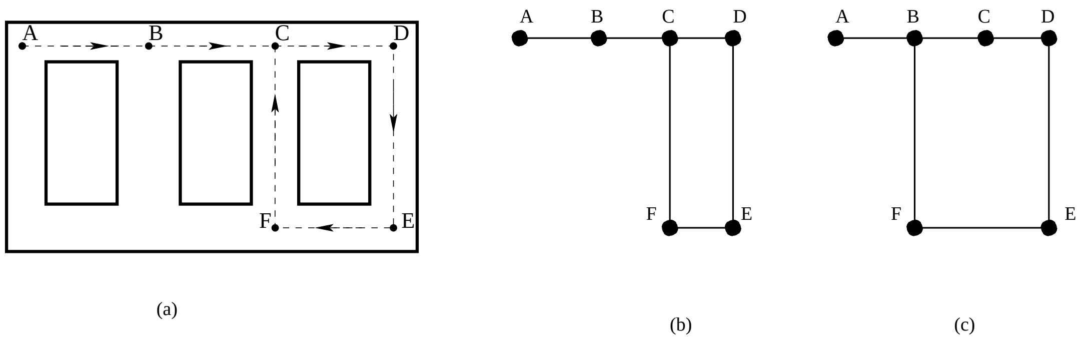


Figure 9.21: (a) The agent goes around the block visiting places $B,\ldots,F,C$ in the order suggested in the figure. Intersections B and C look alike to the agent. (b) and (c) represent two possible topological maps for the environment in (a).

9.4.3 Trace example

We illustrate the topological map building algorithm with the environment of figure 9.21 in which the robot goes around a block and finds itself with two posibilities for the topological map (this is example 19, page 59). For the purpose of the example, *the robot start the navigation at place B instead of place A*.

The set of schemas associated with this environment are defined as follows:[26]

```
;; action definitions
(create-action :name 'ml :type 'travel
               :measurement-type 'interval-measurement)
(create-action :name 'tr :type 'turn
               :measurement-type 'interval-measurement)

;; view definitions

(setf v (create-view 'v))
(setf v1 (create-view 'v1))
(setf v2 (create-view 'v2))
(setf v3 (create-view 'v3))
(setf v4 (create-view 'v4))

;; view-action-view-sequence
```

[26]Here we show the actual input file used to generate the trace.

```
(setf tour-views
  (list v  '(ml (:lb 0.8 :ub 1.2))
        v  '(ml (:lb 0.8 :ub 1.2))
        v1 '(tr (:lb 260 :ub 280))
        v2 '(ml (:lb 0.8 :ub 1.2))
        v1 '(tr (:lb 260 :ub 280))
        v  '(ml (:lb 0.8 :ub 1.2))
        v  '(tr (:lb 260 :ub 280))
        v3 '(ml (:lb 0.8 :ub 1.2))
        v4 '(tr (:lb 260 :ub 280))
        v  '(ml (:lb 0.9 :ub 1.1))   ;;; use a different measurement
        v1))

;; Convert view-action-view sequence to schemas.  By default, the program
;; creates a new distinctive state for each view occurrence in such sequence.

(setf tour-experiences (ssh-create-experiences tour-views))
(ssh-worlds-add-experiences tour-experiences)

;;; invoke the map building function.

(ssh-find-maps)
```

The function **ssh-find-maps** implements the building of the topological map. Explanations of why two objects are proven equal or different are generated by the program. Here is the trace generated for this example.

```
Current partial model PMODEL-0 <NIL -- schemas[0], paths[0], dpaths[0],
                                   places[0], dstates[0]>

 >> Considering experience <(DS-0 V),ML[10],(DS-1 V)>
     PMODEL-0 |= -(DS-0 = DS-1) {dstates distance 10 on dpath DPATH-0}
     PMODEL-0 |= -(PLACE-0 = PLACE-1) {place distance 10 on dpath DPATH-0}
```

In addition to the model name, a list of assumed equalities (NIL in the example above) as well as the number of different SSH objects represented by the partial model are printed. The number of SSH objects in a partial model is used to order the search process according to the circumscription policy associated with $TT(E)$. Figure 9.22 shows the dstates, places and dpaths created by the map building algorithm.[27] The next interesting development occurs when the agent reaches place E and observes $V1$ again.

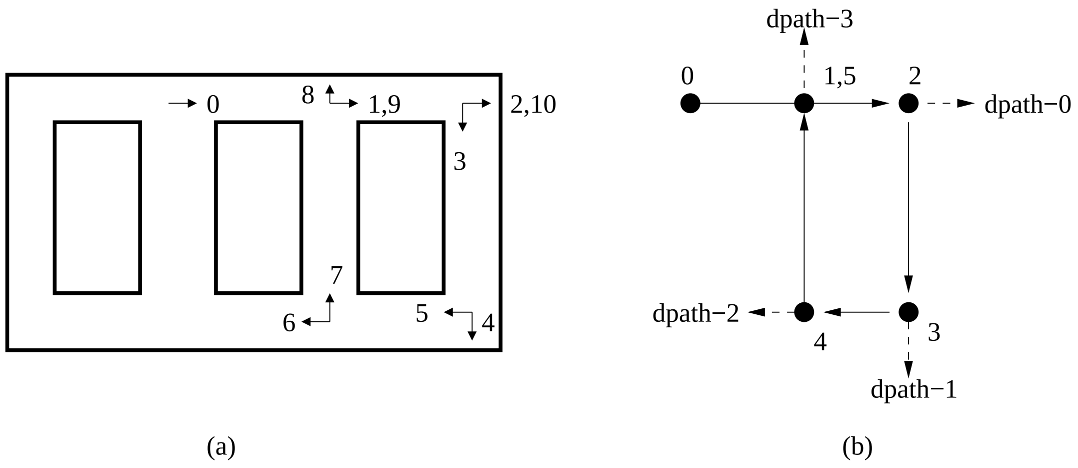


Figure 9.22: Dstates, places and dpaths created by the map building algorithm for the exploration of the environment in figure 9.21. (a) Numbers identify the dstates created by the map building algorithm. (b) Numbers identify places. Notice that PLACE-1 and PLACE-5 are two names for the same place.

```
Current partial model PMODEL-0 <NIL -- schemas[1], paths[0], dpaths[1],
                                  places[2], dstates[2]>

 >> Considering experience <(DS-1 V),ML[10],(DS-2 V1)>
     PMODEL-0 |= -(DS-1 = DS-2) {dstates distance 10 on dpath DPATH-0}
     PMODEL-0 |= -(PLACE-1 = PLACE-2) {place distance 10 on dpath DPATH-0}

Current partial model PMODEL-0 <NIL -- schemas[2], paths[0], dpaths[1],
                                  places[3], dstates[3]>

 >> Considering experience <(DS-2 V1),TR[-90],(DS-3 V2)>
     PMODEL-0 |= -(DS-2 = DS-3) {dstates linked by turn action}
     PMODEL-0 |= -(DPATH-1 = DPATH-0) {dpaths related by turn action}

Current partial model PMODEL-0 <NIL -- schemas[3], paths[0], dpaths[2],
```

[27]In the implementation, *dpaths* represent ordered dstates linked by travel actions. Dpaths correspond to paths that only have one direction associated with them. Paths are created when the agent has traveled in both direction of a path. At that time, two dpaths are associated with the path, one for each path's direction.

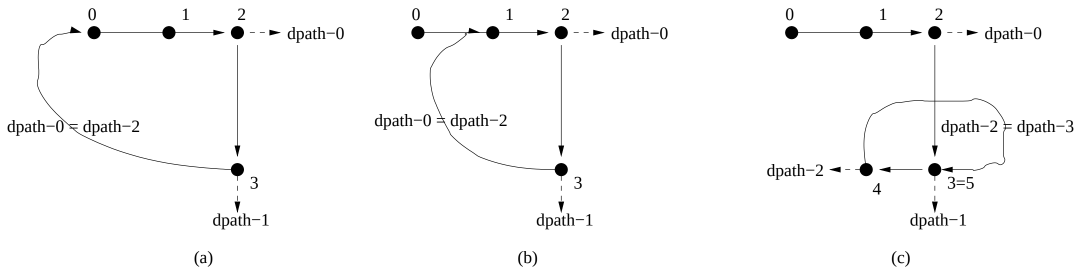


Figure 9.23: **Detecting inconsistent maps by looking ahead when branching**. When the agent reaches place F (figure 9.21), view V is observed again (it was observed at *DS-0*, *DS-1* and *DS-5*, figure 9.22). The maps associated with the equalities $DS-0 \overset{PMODEL-1}{=} DS-6$ and $DS-1 \overset{PMODEL-2}{=} DS-6$ are depicted in (a) and (b). These two maps are detected inconsistent in the look ahead step of our map building algorithm (see figure 9.19).
The second time the agent reaches place C, it turns right, and observes V. Again, the look ahead step detects that the equality $DS-9 \overset{PMODEL-5}{=} DS-5$ leads to the inconsistent map depicted in (c).

```
                                 places[3], dstates[4]>

 >> Considering experience <(DS-3 V2),ML[10],(DS-4 V1)>
   Filtering possible alternatives for  DS-4 : =? (DS-2)
       >> FILTER OUT (DS-4 = DS-2) in PMODEL-0
              {dstates along different dpaths DPATH-1, DPATH-0}

     PMODEL-0 |= -(DS-3 = DS-4) {dstates distance 10 on dpath DPATH-1}
     PMODEL-0 |= -(PLACE-2 = PLACE-3) {place distance 10 on dpath DPATH-1}
```

When the agent reaches place E (see figure 9.21), view $V1$ is observed again (it was observed at *DS-2*). The program concludes that *DS-4* (the current distinctive state) and *DS-2* are different, and consequently, filters out *DS-2* from the list of dstates that can be equal to *DS-4*. The next interesting development occurs when the agent reaches place F and observes V again.

```
Current partial model PMODEL-0 <NIL -- schemas[4], paths[0], dpaths[2],
                                 places[4], dstates[5]>

 >> Considering experience <(DS-4 V1),TR[-90],(DS-5 V)>
   Filtering possible alternatives for  DS-5 : =? (DS-0 DS-1)
```

```
     PMODEL-0 |= -(PLACE-3 = PLACE-0)
       {PLACE-3 is to the right of DPATH-0, PLACE-0 is on dpath DPATH-0}
     >> FILTER OUT (DS-5 = DS-0) in PMODEL-0
            {dstates at different places PLACE-3, PLACE-0}
     PMODEL-0 |= -(PLACE-3 = PLACE-1)
       {PLACE-3 is to the right of DPATH-0, PLACE-1 is on dpath DPATH-0}
     >> FILTER OUT (DS-5 = DS-1) in PMODEL-0
            {dstates at different places PLACE-3, PLACE-1}
     PMODEL-0 |= -(DS-4 = DS-5) {dstates linked by turn action}
     PMODEL-0 |= -(DPATH-2 = DPATH-1) {dpaths related by turn action}

Current partial model PMODEL-0 <NIL -- schemas[5], paths[0], dpaths[3],
                                  places[4], dstates[6]>

 >> Considering experience <(DS-5 V),ML[10],(DS-6 V)>
   Filtering possible alternatives for  DS-6 : =? (DS-0 DS-1 DS-5)
     >> FILTER OUT (DS-6 = DS-5) in PMODEL-0
            {DS-5 is before DS-6 in dpath  DPATH-2}
 >> Creating possible extensions

   - PMODEL-0 + {DS-0 = DS-6} --> PMODEL-1
      PMODEL-1 |=  DPATH-0 = DPATH-2  {dstate DS-0 along both dpaths}
      PMODEL-1 |=  DPATH-2 = DPATH-0  {dstate DS-5 along both dpaths}
      PMODEL-1 |= -(DPATH-0 = DPATH-2)
            {PLACE-3 is to the right of DPATH-0 and on dpath DPATH-2}

      LOOK AHEAD -->  PMODEL-1 **** INCONSISTENT ****


   - PMODEL-0 + {DS-1 = DS-6} --> PMODEL-2
      PMODEL-2 |=  DPATH-0 = DPATH-2  {dstate DS-1 along both dpaths}
      PMODEL-2 |=  DPATH-2 = DPATH-0  {dstate DS-5 along both dpaths}
      PMODEL-2 |= -(DPATH-0 = DPATH-2)
            {PLACE-3 is to the right of DPATH-0 and on dpath DPATH-2}

      LOOK AHEAD -->  PMODEL-2 **** INCONSISTENT ****

      PMODEL-0 |= -(PLACE-3 = PLACE-4) {place distance 10 on dpath DPATH-2}
```

When the agent reaches place *F* (see figure 9.21), view *V* is observed again (it was observed at *DS-0*, *DS-1* and *DS-5*) at dstate *DS-6*. The program deduces that *DS-6* is different from *DS-5*. Two extensions of *PMODEL-0* are created to explore the possibilities

$DS-6 \overset{teq}{=} DS-0$ and $DS-6 \overset{teq}{=} DS-1$. The look ahead step right away concludes that these equalities render the corresponding extensions inconsistent, and consequently, *DS-6* is different from the previously seen dstates (see figure 9.23). *PMODEL-0* continues to be the only possible model. The next interesting development occurs when the agent reaches place C, turns right, and observes view V again.

```
Current partial model PMODEL-0 <NIL -- schemas[6], paths[0], dpaths[3],
                                  places[5], dstates[7]>

 >> Considering experience <(DS-6 V),TR[-90],(DS-7 V3)>
     PMODEL-0 |= -(DS-6 = DS-7) {dstates linked by turn action}
     PMODEL-0 |= -(DPATH-3 = DPATH-2) {dpaths related by turn action}

Current partial model PMODEL-0 <NIL -- schemas[7], paths[0], dpaths[4],
                                  places[5], dstates[8]>

 >> Considering experience <(DS-7 V3),ML[10],(DS-8 V4)>
     PMODEL-0 |= -(DS-7 = DS-8) {dstates distance 10 on dpath DPATH-3}
     PMODEL-0 |= -(PLACE-4 = PLACE-5) {place distance 10 on dpath DPATH-3}

Current partial model PMODEL-0 <NIL -- schemas[8], paths[0], dpaths[4],
                                  places[6], dstates[9]>

 >> Considering experience <(DS-8 V4),TR[-90],(DS-9 V)>
   Filtering possible alternatives for  DS-9 : =? (DS-0 DS-1 DS-5 DS-6)
     >> FILTER OUT (DS-9 = DS-6) in PMODEL-0
            {dstates at different places PLACE-5, PLACE-4}
 >> Creating possible extensions

  - PMODEL-0 + {DS-0 = DS-9} --> PMODEL-3
     PMODEL-3 |= -(DS-8 = DS-0) {dstates linked by turn action}
     PMODEL-3 |= -(DPATH-0 = DPATH-3) {dpaths related by turn action}
     PMODEL-3 |= -(PLACE-5 = PLACE-1) {place distance 10 on dpath DPATH-0}
     PMODEL-3 |= PLACE-0 = PLACE-5  {dstate DS-0 at both places}

  - PMODEL-0 + {DS-1 = DS-9} --> PMODEL-4
     PMODEL-4 |= -(DS-8 = DS-1) {dstates linked by turn action}
     PMODEL-4 |= -(DPATH-0 = DPATH-3) {dpaths related by turn action}
     PMODEL-4 |= -(PLACE-5 = PLACE-2) {place distance 10 on dpath DPATH-0}
     PMODEL-4 |= -(PLACE-0 = PLACE-5) {place distance 10 on dpath DPATH-0}
     PMODEL-4 |= PLACE-1 = PLACE-5  {dstate DS-1 at both places}
  - PMODEL-0 + {DS-5 = DS-9} --> PMODEL-5
     PMODEL-5 |= -(DS-8 = DS-5) {dstates linked by turn action}
     PMODEL-5 |= PLACE-3 = PLACE-5  {dstate DS-5 at both places}
     PMODEL-5 |= -(PLACE-5 = PLACE-3)
       {PLACE-5 is to the right of DPATH-2, PLACE-3 is on dpath DPATH-2}

     LOOK AHEAD -->  PMODEL-5 **** INCONSISTENT ****


  - PMODEL-0 + {DS-9 = DS-9} --> PMODEL-6
     PMODEL-6 |= -(DS-8 = DS-9) {dstates linked by turn action}
     PMODEL-6 |= -(DPATH-4 = DPATH-3) {dpaths related by turn action}

PMODEL-0 ***** EXPLORED *****
```

When the agent reaches place *C* again (see figure 9.21), it turns right, and observes V. In this situation, the current distinctive state *DS-9* can be equal to *DS-0*, equal to *DS-1* or a new distinctive state (see figure 9.23 for why *DS-9*$\neq$*DS-5*). Three new extensions are created, *PMODEL-3*, *PMODEL-4*, *PMODEL-6*, to explore these alternatives. *PMODEL-3* and *PMODEL-4* correspond to the maps (c) and (b) in figure 9.21. Given the set of experiences explained up to this point, *PMODEL-6* is not a minimal topological map for these experiences. However, the program generates this alternative in case the other two eventually fail. *PMODEL-0* becomes explored as some extensions has been created. The search continue by picking a minimal model, in this case either *PMODEL-3* or *PMODEL-4*.

```
Current partial model PMODEL-3 <(DS-0 DS-9) -- schemas[9], paths[0],
                                         dpaths[4], places[5], dstates[9]>

 >> Considering experience <(DS-9 V),ML[10],(DS-10 V1)>
     PMODEL-3 |= -(DS-1 = DS-10) {dstates have different views.}

     PMODEL-3 |= (DS-10 = DS-1)
           {deterministic actions: <(DS-9 V),ML[10],(DS-10 V1)> ,
                                   <(DS-0 V),ML[10],(DS-1 V)>}

PMODEL-3 <(DS-0 DS-9) -- schemas[9], paths[0], dpaths[4],
                          places[5], dstates[9]> **** INCONSISTENT ****
```

Once the agent travels from place C to place D again (see figure 9.21), *PMODEL-3* becomes inconsistent. This is the case since in this model $DS-0 \overset{teq}{=} DS-9$ and so after performing ml in $DS-0$ the map predicts view V but the actual experience renders view $V1$. This is just the assumption that actions are deterministic. The algorithm then considers *PMODEL-4*, which successfully explain the schema $\langle\,(DS-9\,V),\,ML[10],\,(DS-10\,V1)\,\rangle$ and becomes a minimal map since no more experiences must be explained. *PMODEL-4* is depicted in figures 9.21b and 9.22b.

```
Current partial model PMODEL-4 <(DS-1 DS-9) -- schemas[9], paths[0],
                                          dpaths[4], places[5], dstates[9]>

 >> Considering experience <(DS-9 V),ML[10],(DS-10 V1)>

     PMODEL-4 |= (DS-10 = DS-2)
           {deterministic actions: <(DS-9 V),ML[10],(DS-10 V1)> ,
                                   <(DS-1 V),ML[10],(DS-2 V1)>}

PMODEL-4 **** IS A MINIMAL MAP  ****
```

9.5 Map building examples

In this section we describe two examples where a physical robot (Vulcan) builds a SSH map. The purpose of these examples is to illustrate how the concepts in the previous section apply in office like environments. The first example (Section 9.5.1), shows how the agent using the concept of boundary regions can distinguish otherwise identical distinctive states. In the second example (Section 9.5.2), Vulcan explores Taylor's second floor defining distinctive states at the different corridor intersections.

9.5.1 Rectangular environment

This section shows the SSH map learned by the wheelchair (Vulcan) while exploring a symmetric environment (namely, a rectangular room). Symmetric environments are particularly difficult to handle as the same view might occur at different distinctive states, and the SSH causal level is not enough to distinguish some distinctive states. By using topological and metrical information the agent can distinguish the different places of the environment.

The environment to explore is a 3m $\times$ 8m rectangle, as illustrated in Figure 9.24. The exploration strategy was set to execute a forward action when possible, otherwise turn right, and stop when back to a previously visited distinctive state. Notice that the exploration strategy is defined in terms of the causal language actions *Forward* and *Turn Right*. The robot chose the most appropriate control law associated with the causal command, which in this case were *follow left wall* and *align to the right w.r.t. the front wall* respectively. As illustrated in Figure 9.24, different distinctive states look alike (i.e. share the same view). For example, distinctive states *DS-1* and *DS-5* share the same view *VIEW-1*.

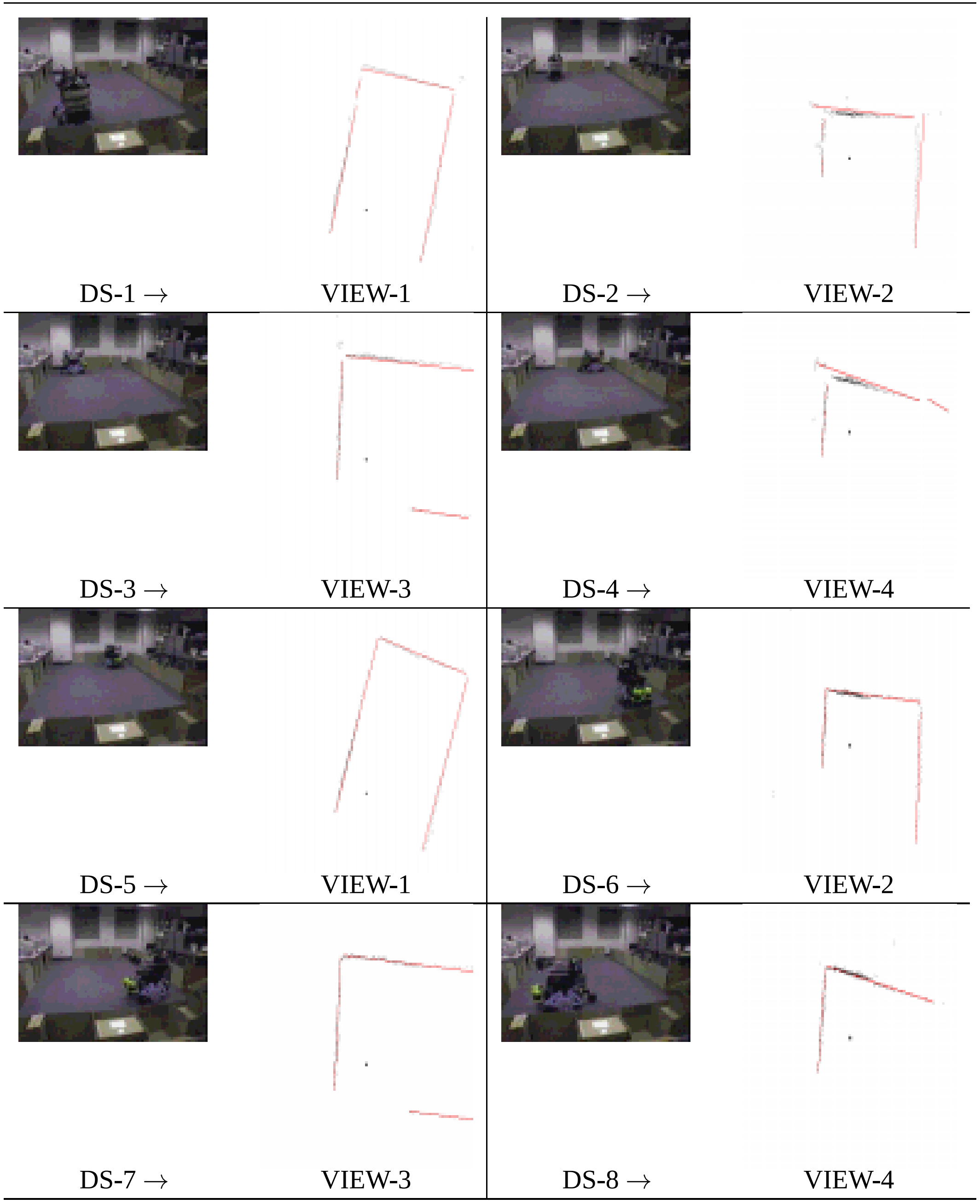


Figure 9.24: **Rectangular room exploration**. For each visited distinctive state the figure shows its corresponding view. The same view occurs at different corners. For instance, *VIEW-1* occurs at distinctive states *DS-1*, *DS-5* and *DS-8*. Using topological and local metrical information, the robots concludes that *DS-1*≠*DS-5* and *DS-1*=*DS-8*.

Next we provide a commented trace of the different steps and reasoning the robot used during the exploration. At each step the robot continues the exploration by performing an action (forward or turn right) and identifying a distinctive state once the action is performed.

Initially, the robot starts at one of the corners of the environment and creates a distinctive state. Here is the program output associated with this step:

```
-----------------------------
Finding current place.
  Finding view.
  No matching view.
    + Creating new view.
    > View found: VIEW-1.
  Finding distinctive state.
    + Creating new d-state DS-1.
  > Distinctive state found: DS-1.
> Place found: A.
-----------------------------
```

Since at this point the robot has no stored views, it gives a new name (*VIEW-1*) to its current view. A new distinctive state (*DS-1*) is created and associated with this view. The place at which this distinctive state is located is created and called *A* (see Axiom 5.16, page 41).

```
-----------------------------
Performing action: Forward.
  Control law: Follow left wall
   Executing Follow left wall ... Done.
-----------------------------
Finding current place.
  Finding view.
  No matching view.
    + Creating new view.
    > View found: VIEW-2.
  Finding distinctive state.
    + Creating new d-state DS-2.
  > Distinctive state found: DS-2.
> Place found: B.
-----------------------------
```

The view observed at this point does not match a known view and consequently a new one is created. Since the view is new, so is the current distinctive state. This is the case since a distinctive state has associated a unique view (Axiom 4.7). A new place *B* is created since the robot executed a travel action which by definition changes the place the robot is at (Corollary 1, page 46).

```
-----------------------------
Performing action: Right.
  Control law: Turn right (ref = front wall)
    Executing Turn right (ref = front wall) ... Done.
-----------------------------
Finding current place.
  Finding view.
  No matching view.
    + Creating new view.
    > View found: VIEW-3.
  Finding distinctive state.
    + Creating new d-state DS-3.
  > Distinctive state found: DS-3.
> Place found: B.
-----------------------------
```

Notice that since the robot performed a *Turn* action, it is still at the same place *B* (Axiom 5.17), although at a different distinctive state *DS-3*.

```
-----------------------------
Performing action: Forward.
  Control law: Follow left wall
   Executing Follow left wall ... Done.
-----------------------------
Finding current place.
  Finding view.
  No matching view.
    + Creating new view.
    > View found: VIEW-4.
  Finding distinctive state.
    + Creating new d-state DS-4.
  > Distinctive state found: DS-4.
> Place found: C.
-----------------------------
```

Since the robot performed a *Travel* action, it is at different place from *B*. Since the view is new, it creates a new distinctive state and a new place *C*. The robot does not check whether it is back at *A*, since it has not experienced the current view at place *A*.

```
-----------------------------
Performing action: Right.
  Control law: Turn right (ref = front wall)
    Executing Turn right ... Done.
-----------------------------
Finding current place.
  Finding view.
  Views matching with score > 0.7:
      1. VIEW-1 (score = 0.8)
    > View found: VIEW-1.
  Finding distinctive state.
    D-states associated with VIEW-1:
      1. DS-1
    - Topology excludes these d-states:
      1. DS-1   (right of current path)
    + Creating new d-state DS-5.
  > Distinctive state found: DS-5.
> Place found: C.
-----------------------------
```

At the current physical location the robot environment looks as in VIEW-1. The only known distinctive state associated with *VIEW-1* is *DS-1*, which is at place *A*. The robot determines that the current place (*C*) is to the right of the path (boundary) from place *A* to place *B*. Consequently, the robot cannot be at place *A*, and so it cannot be at *DS-1*. A new distinctive state *DS-5* is then created.

```
-----------------------------
Performing action: Forward.
  Control law: Follow left wall
   Executing Follow left wall ... Done.
-----------------------------
Finding view.
  Views matching with score > 0.7:
      1. VIEW-2 (score = 0.8)
    > View found: VIEW-2.
  Finding distinctive state.
    D-states associated with VIEW-2:
      1. DS-2
    - Topology excludes these d-states:
      1. DS-2   (right of current path)
    + Creating new d-state DS-6.
  > Distinctive state found: DS-6.
> Place found: D.
-----------------------------
```

By a reasoning similar to the one used to distinguish *DS-1* from *DS-5*, the robot can distinguish the current distinctive state (*DS-6*) from *DS-2*. Notice that this time the robot knows that it is not at place *C* (since it just traveled from it, lemma 1) and it is not at place *B*, since the current place (*D*) is on the right of the path from *B* to *C*.

```
-----------------------------
Performing action: Right.
  Control law: Turn right (ref = front wall)
   Executing Turn right (ref = front wall) ... Done.
-----------------------------
Finding current place.
  Finding view.
    Views matching with score > 0.7:
      1. VIEW-3 (score = 0.799264)
    > View found: VIEW-3.
  Finding distinctive state.
    D-states associated with VIEW-3:
      1. DS-3
    - Topology excludes these d-states:
      1. DS-3   (right of boundary)
    + Creating new d-state DS-7.
  > Distinctive state found: DS-7.
> Place found: D.

-----------------------------
Performing action: Forward.
  Control law: Follow left wall
   Executing Follow left wall ... Done.
-----------------------------
Finding current place.
  Finding view.
    Views matching with score > 0.7:
      1. VIEW-4 (score = 0.710665)
    > View found: VIEW-4.
  Finding distinctive state.
    D-states associated with VIEW-4:
      1. DS-4
    - Topology excludes these d-states:
      1. DS-4   (right of current path)
    + Creating new d-state DS-8.
  > Distinctive state found: DS-8.
> Place found: E
-----------------------------
```

At the current physical location the robot environment looks similar to VIEW-4. The only known distinctive state associated with *VIEW-4* is *DS-4*. A similar argument that allows the agent to differentiate *DS-3* from *DS-4* allows it to differentiate *DS-4* from *DS-8*. Notice that the agent does not know that it is back to place *A*, and it creates a new place for *DS-8*, place *E*.

```
-----------------------------
Performing action: Right.
  Control law: Turn right (ref = front wall)
   Executing Turn right (ref = front wall) ... Done.
-----------------------------
Finding current place.
  Finding view.
    Views matching with score > 0.7:
      1. VIEW-1 (score = 0.742667)
    > View found: VIEW-1.
  Finding distinctive state.
    D-states associated with VIEW-1:
      1. DS-1
      2. DS-5
    - Topology excludes these d-states:
      1. DS-5   (right of current path)
    + Geometry is compatible with:
      1. DS-1 at 0.796978 m., 18.8011 deg.
  > Distinctive state found: DS-1.
> Place found: A.
-----------------------------
```

Since the topology cannot rule out the possibility of being at *DS-1*, the agent uses metrical information to check whether it might be back to *DS-1*. In order to do so, it finds a path from *DS-01* to the current distinctive state, and creates a two dimensional frame of reference with the places on that path. It finds that the current distinctive state is $0.79m$ apart from *DS-01* and that its direction to *DS-01* is about $18deg$. Given our current metrical error tolerance, the robot accepts the hypothesis that it is back to *DS-1*. Should the agent not rely on metrical information, the hypothesis of being back to *DS-1* will be accepted right away since it is consistent with our minimality criteria when building the topological map.

9.5.2 Taylor's Second floor

In this experiment we illustrate the map building process while navigating the second floor of Taylor Hall. Figure 9.25 shows a footprint of this environment with the different distinctive states identified after exploration. Figure 9.26 shows the views associated with the distinctive states in this environment. Table 9.27 shows the view matching values associated with the different distinctive states in Taylor 2nd floor.

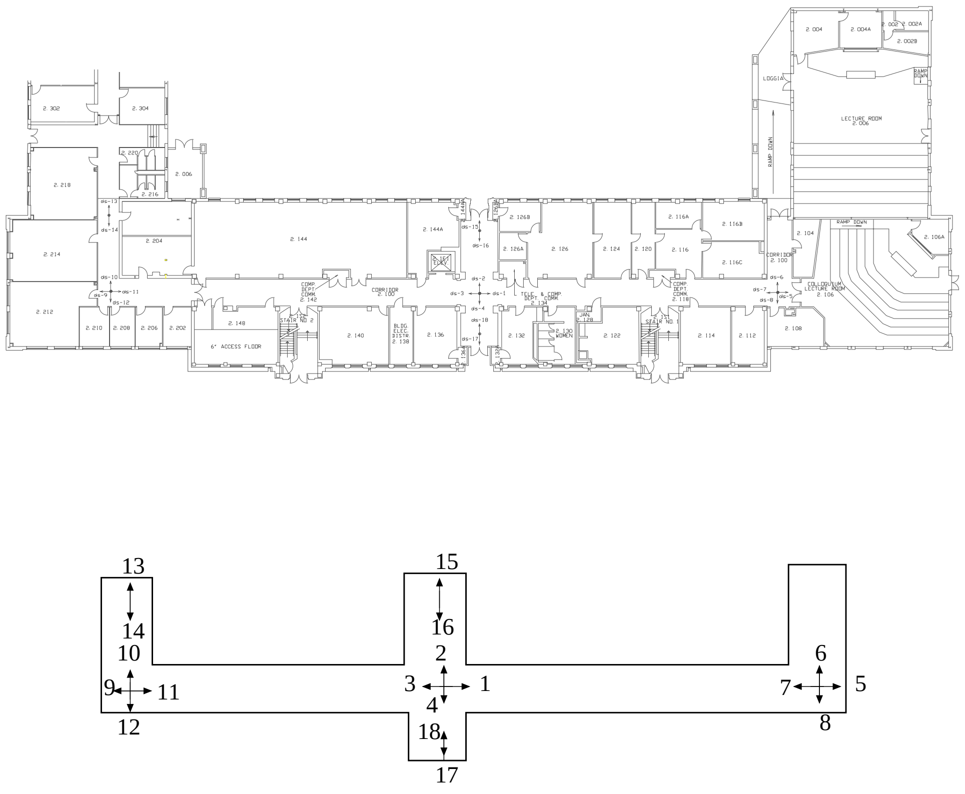


Figure 9.25: Footprint of Taylor's second floor and the distinctive states found while exploring this environment.

The exploration strategy for the environment is such that when arriving to a distinctive state, the robot checks whether it has been there before. If it has, then it checks what action has not been performed and performs it. If the distinctive state is new, the robot rotates 360^o left identifying other distinctive states in the same place. If a trajectory

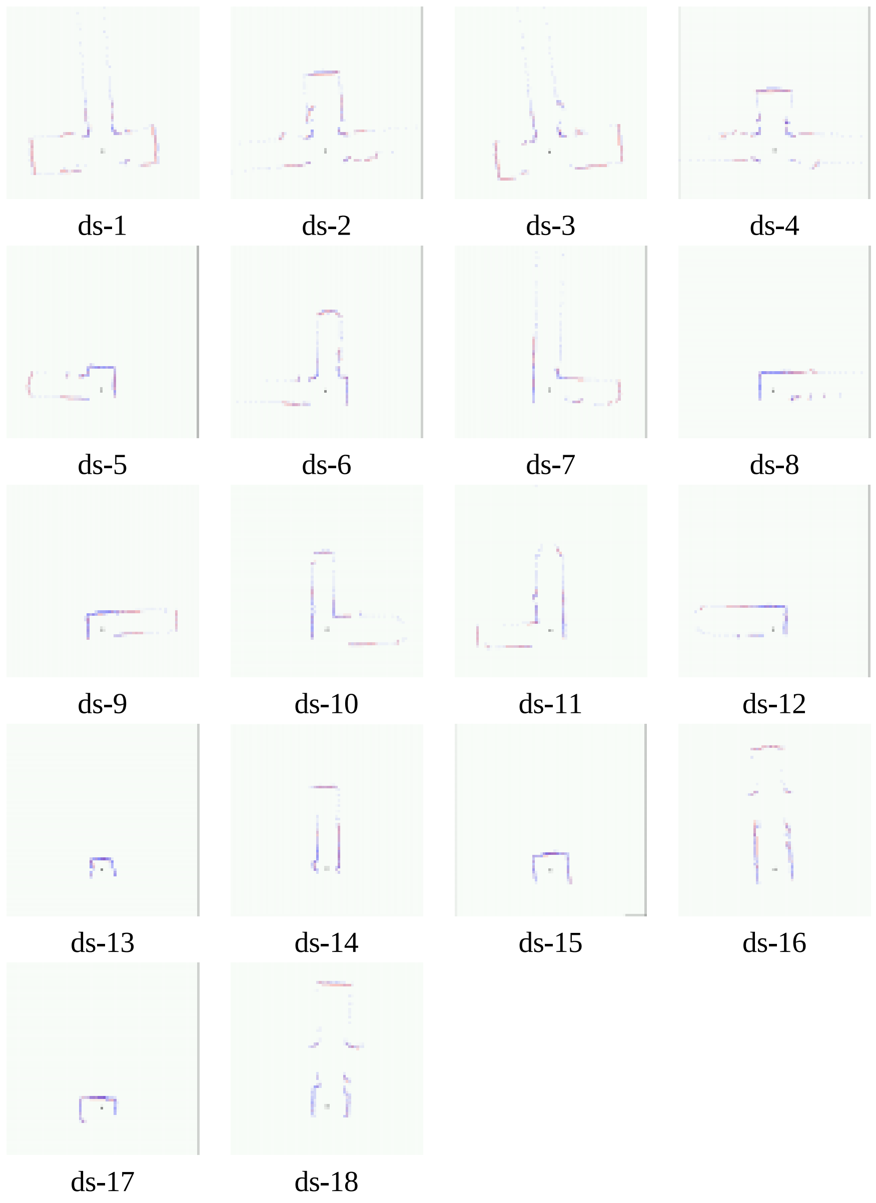


Figure 9.26: Views associated with the distinctive states in Taylor 2nd floor.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	100	7	44	6	0	0	0	13	7	8	12	11	0	39	9	0	14	9
2	7	100	4	3	17	24	24	29	23	0	12	11	29	23	37	23	37	42
3	44	4	100	4	18	0	0	0	0	21	5	0	7	39	5	4	5	0
4	6	30	4	100	0	21	23	38	6	0	0	13	0	0	30	21	42	40
5	0	17	18	0	100	0	0	0	0	15	23	25	23	27	25	0	0	17
6	0	24	0	21	0	100	0	0	15	12	8	0	0	14	34	16	26	14
7	0	24	0	23	0	0	100	0	25	35	0	0	18	0	0	33	55	17
8	13	29	0	38	0	0	0	100	3	0	9	42	43	27	47	22	58	26
9	7	23	0	6	0	15	25	36	100	8	15	0	20	21	42	24	37	22
10	8	0	21	0	15	12	35	0	8	100	7	21	13	46	17	16	0	0
11	12	12	5	0	23	8	0	9	15	7	100	0	13	7	10	18	7	7
12	11	11	0	13	25	0	0	42	0	21	0	100	0	20	23	0	0	17
13	0	29	7	0	23	0	18	43	20	13	13	0	100	0	32	19	35	0
14	39	23	39	0	27	14	0	27	21	46	7	20	0	100	19	19	0	26
15	9	37	5	30	25	34	0	47	42	17	10	23	32	19	100	0	90	15
16	0	23	4	21	0	16	33	22	24	16	18	0	19	19	0	100	14	21
17	14	37	5	42	0	26	55	58	37	0	7	0	35	0	90	14	100	14
18	9	42	0	40	17	14	17	26	22	0	7	17	0	26	15	21	14	100

Figure 9.27: Similarity values among the different views associated with the distinctive states identified in Taylor's second floor. Scale is 0 to 100.

following strategy can be applied, the robot does so. Otherwise it rotates right to the closest distinctive state, where the procedure is repeated. Using this strategy the order in which distinctive states are visited for the environment of figure 9.25 is summarized in figure 9.28.

For the experiment we used a threshold value of 0.6 in order to decide whether two list of segments correspond to the same view. This value is consistent with the intraclass similarity value provided by our view matcher, as discussed in example 31 (page 140). Accordingly, in the Taylor second floor environment the robot experiences sensory aliasing for distinctive states ds-15 and ds-17 (see figures 9.26 and 9.27). However, these distinctive states are along the same path at different places, and topological information is enough to distinguish them (see figure 9.28).

Actions	Distinctive states	Explanation
Reach ds-1 and turn in place	1, 2, 3, 4, 1	
Follow corridor until ds-5 Rotate in place	5, 6, 7, 8, 5	ds-5 has a view not seen so far.
Rotate right until follow corridor is applicable.	8,7	
Follow corridor to ds-3	3	view at ds-3 has been seen before at only one distinctive state. The experiences are consistent with being at ds-3
Follow corridor to ds-9 Rotate in place Rotate right until follow corridor is applicable.	9,10,11,12, 9,10	ds-9 to ds-12 have views not seen so far.
Follow corridor until ds-13	13,14,13	ds-13 and ds-14 have views not seen so far.
Rotate right until follow corridor is applicable.	14	
Follow corridor to ds-12.	12	view at ds-12 has been seen before at only one distinctive state. The experiences are consistent with being at ds-12.
Rotate left until follow corridor is applicable.	11	Since the robot already turned right at ds-12,it nows turns left.
Follow corridor to ds-1		view at ds-1 has been seen before at only one distinctive state. The experiences are consistent with being at ds-1.
Rotate left until follow corridor is applicable.	2	Since the robot already followed the corridor from ds-1,it nows turns left.
Follow corridor to ds-15. Rotate in place. Rotate right until follow corridor is applicable.	15,16,15,16	ds-15 has a view not seen so far.
Follow corridor to ds-4		view at ds-4 has been seen before at only one distinctive state. The experiences are consistent with being at ds-4.
Follow corridor to ds-17	17	view at ds-17 has been seen before (at ds-15). Topological information indicates that the robot is not at ds-15, since the place of ds-15 is before the current place in the current path.
Rotate in place Follow corridor to ds-2	18,17,18,2	view at ds-2 has been seen before at only one distinctive state. The experiences are consistent with being at ds-2.

Figure 9.28: Summary of exploration for Taylor's second hall.

9.6 Summary

We have presented the main issues of a particular implementation of the SSH using Vulcan. At the SSH control level, an observer architecture is used to implement trajectory following and hill climbing control laws. The same control law (equation 9.1, page 130) used in conjunction with different *model trackers* (figure 9.3, page 131) produces most of Vulcan's behaviors (for instance, follow a corridor, go through doorways, hill climb to a corridor intersection, etc.). The implementation satisfies the SSH control level guarantees: the robot behaviors start and terminate in the neighborhood of the fixed points associated with hill climbing strategies. While the focus of this dissertation has not been on the SSH control level, it is worthwhile noticing that in general implementing the control level in a physical robot is half of the battle when implementing the SSH.

Views at the causal level were defined in terms of segments extracted from a laser scan at the particular robot location (section 9.3, page 138). The results in example 31 and section 9.5.2 (pages 142 and 178) show that this method is adequate in office like environments: the views there defined have a well defined discriminant boundary among the different view classes identified by the robot. Our method uses one laser scan to define a view. Other methods exist where more than one laser scan (or multiple sensors) are used to define a view ([Duckett and Nehmzow, 2000]). No attempt has been done to compare our method to these other ones.

The implementation of the causal, topological and metrical levels has focussed on identifying and keeping track of the different maps consistent with the agent's experiences. A logic program derived from the circumscriptive theory defined in chapter 4 was defined for calculating the causal models associated with a set of experiences (section 9.4.1, page 147). The same technique unfortunately does not scale up when building the topological map. The problem of building a topological map was stated then as a *"best first"* search problem. The states of the search correspond to partial models of $TT(E)$. At each step of the search a schema $\langle\, ds, a, ds' \,\rangle$ has to be explained. Either the identity of ds' can be proved or a search branch is created for every previously known distinctive state ds'' that cannot be proven to be different from ds'. In each branch the assumption $ds'' \overset{teq}{=} ds'$ will be the case. The next state to explore is the one that is minimum according to the order associated with the circumscription policy for $TT(E)$. This search algorithm was described in section 9.4.2 (page 152).

While not explicitly described in this chapter, the robot exploration strategy is such that should more than one map be consistent with the robot experiences, the robot tries immediately to distinguish among the different alternatives (i.e. the robot performs the rehearsal procedure). The knowledge base is such that one could query for a sequence of actions (and so observations) that will distinguish among the possible map alternatives.

Metrical information has been used to *refute* a given topological map by showing that it is impossible to project the map into a two dimensional space preserving the estimated distance and relative orientation between consecutive places in a path (chapter 7, page 90). A more sophisticated use of metrical information has not been explored in the implementation (for instance, occupancy models associated with a place has not been implemented). Sections 9.4.3 (page 161) and 9.5.1 (page 169) illustrate the use of metrical information while building the topological map.

Chapter 10

Related Work

In this section we review some of the relevant literature in the different areas comprising this work. These areas include: cognitive robotics, qualitative reasoning, robotics and map building.

10.1 Cognitive theories of space representation

Human knowledge of large-scale space is sometimes called the *cognitive map*.[1] The cognitive map serves two functions with regard to wayfinding: representing environments, and the corresponding ability to use the representation to move from place to place within the mapped environment [Kortenkamp *et al.*, 1995]. Among the main characteristics of the cognitive map we have: the use of multiple frames of reference, qualitative representation of metrical information, and connectivity relations among landmarks. Next we review the computational theories of the cognitive map that are closest to our current work. Some other systems can be found in [Leiser and Zilbershatz, 1989, Gopal *et al.*, 1989, Gopal and Smith, 1990, O'Neill, 1991, Engelson and McDermott, 1992a, Yeap, 1988].

10.1.1 PLAN

Kortenkamp et al. [Kortenkamp *et al.*, 1995] have developed an integrated representation of large-scale space called PLAN (Prototypes, Location and Associative Networks). In this

[1]Large-scale space is defined as space whose structure is at a much larger scale than the sensory horizon of the agent. Thus, to learn a map, the agent must travel through the space, gathering local observations and inferring their global relationships from the actions linking them [Kuipers, 2000].

theory, landmarks function as a kind of environmental index. Recognizing nearby landmarks is enough to tell one where one is in a familiar environment. Objects (landmarks) are represented by *prototypes*.[2] A network of landmarks is created as the agent travels the environment. In this network, nodes represent landmarks and edges between nodes represent the agent's ability to go between two proximate landmarks [J., 1991, Schölkopf and Mallot, 1995]. Edges in the network have variable strength values associated with them.[3]

Directional space is represented by a *local map*. A local map provides the relative change in orientation for any neighboring target landmark. A local map is represented by a combination of a 2-D grid and a sectoring information with respect to a normalized viewpoint.[4] Local maps are created at *gateways*: places were a route choice point had been reached and a new landmarks could be seen. A network of local maps is then created associated with the corresponding network of landmarks.

10.1.2 McDermott and Davis

McDermott and Davis have developed a theory where objects other than paths are represented in the cognitive map [McDermott, 1980, McDermott and Davis, 1984, Davis, 1993]. In addition to topological information, a "fuzzy map" is created for representing metrical information. A fuzzy map captures facts about objects by recording their relative positions, orientations, and scales in convenient frames of reference. Uncertainty in metrical information is represented by numerical intervals. Assimilation of new information might constrain the uncertainty on metrical information. Hill climbing or Monte Carlo techniques are used for propagating metrical constraints [McDermott and Davis, 1984].

The MERCATOR program [Davis, 1993] constructs a cognitive map from a sequence of scene descriptions. Objects are represented by sets of polygons. The relative positions of objects are determined by connecting edges. The space representation accounts for uncertainty on position and distance between objects, multiple shape description of the same object, and representation of partial knowledge about objects spatial properties. In addition, operations on the map are justified by a formal semantics. In Appendix G we describe MERCATOR's ontology as well as its associated semantics.

[2]Prototypes are generalizations derived from a range of experience. In such a generalization the features that occur most often come to represent the prototype, whereas features that occur less often are weaker, giving the prototype a statistical nature, reflecting experience [Kortenkamp *et al.*, 1995].

[3]Using fixed links to connect adjacent landmarks yields an inadequate model of human wayfinding. Familiar routes are naturally easier to remember; routes that have only been traversed once or twice are going to be difficult to re-create [Kortenkamp *et al.*, 1995].

[4]The representation of orientation information is similar to the one used by Hernandez [Hernandez, 1994].

10.2 Cognitive Robotics

The area of Cognitive Robotics studies the knowledge representation and reasoning problems faced by an autonomous agent in a dynamic and incompletely known world. The goal is to develop an understanding of the relationship between the knowledge, the perception, and the action of such agent. With respect to robotics, the goal is a *high level robotic control*: develop a system that is capable of generating actions in the world that are appropriate as a function of some current set of beliefs and desires. Most of the work in this area aims for a logical account for the sort of high level cognitive skills listed above. Robots whose design is based on logical representation have the virtue of producing behavior which can be accounted for in terms of *correct reasoning* and *correct representation* [Sandewall, 1996]. Logics for reasoning about actions and planning are used in order to describe the actions of the robots as well as its environment [Shanahan, 1997b, Sandewall, 1994]. Formal specifications are then used as a base to program robots [Lesperance *et al.*, 1994, Sandewall, 1997, Shanahan, 1998]. Next we summarize two of the main works in the area: the work done at the University of Toronto (by Reiter and et al.) and the work done by Murray Shanahan.

10.2.1 GOLOG

The Cognitive Robotics project at the University of Toronto is concerned with endowing robotic or software agents with higher level cognitive functions that involve reasoning, for example, about goals, perception, actions, the mental states of other agents, collaborative task execution, etc. To do this, they describe, in a language suitable for automated reasoning (GOLOG), enough of the properties of the robot, its abilities, and its environment, to permit it to make high-level decisions about how to act. The situation calculus [McCarthy and Hayes, 1969] has been taken as the underlying language for reasoning about the prerequisites and effects of actions, perception, knowledge-producing actions, natural events and actions by other agents [Reiter, 1996]. These methods have been incorporated into a logic programming language for agents called GOLOG (alGOl in LOGic).

GOLOG is a high-level programming language suitable for declaratively defining complex behaviors. In GOLOG [Lesperance *et al.*, 1994],

> "The user provides a specification of the robot's basic actions (their preconditions and effects on the environment) as well as of relevant aspects of the

environment, in an extended version of the situation calculus. He can then specify robot behaviors in terms of these actions in a programming language that allows references to world conditions (e.g. **if** $\exists c(pop\_can(c) \wedge on\_table(c))$ **then** $pick\_up(c)$). The programs can be executed to drive the robot. The interpreter automatically maintains the world model required to execute programs based on the specification. The theoretical framework includes a solution to the frame problem and is very general (it handles dynmaic and incompletely known worlds, as well as perception actions). Given this kind of domain specification, it is also possible to support more sophisticated reasoning, such as task planning at run-time. The specification can also be used to prove the robot control programs correct".

Recently, GOLOG has been extended to CONGOLOG (CONcurrent GOLOG) [De Giacomo *et al.*, 1997] which includes facilities for concurrent execution, interrupting the execution when certain conditions become true, and dealing with exogenous actions.

10.2.2 Shanahan's Work

Shanahan [Shanahan, 1996, Shanahan, 1997a] proposes a logic-based framework in which a robot constructs a model of the world through an abductive process whereby sensor data is explained by hypothesising the existence, locations, and shapes of objects. In [Shanahan, 1998] this work is augmented by an account for *planning* as well as an implementation of these ideas via an abductive logic programming metainterpreter. Next we summarize the main points of these formalizations.

Given a stream of sensor data, represented as the conjunction Υ of a set of observation sentences, the task is to find an explanation of Υ in the form of a logical description (a map) Δ_M of the initial locations and shapes of a number of objects, such that,

$$KB \wedge \Delta_M \models \Upsilon$$

where KB is a theory comprising axioms for change, action, space, relations between movements of objects and robot's sensor data, the movements executed by the robot, etc.[5]

The **semantics** of an agent's space representation, M, is a formula, $\Delta(M)$, in the language of KB, such that $\Delta(M)$ postulates the initial locations and shapes of a number of objects. As an agent travels through its environment, it executes a set of actions, A, and experiences

[5]KB is expressed in the circumpscritive event calculus [Shanahan, 1997b].

a set of observations Υ. As a result, the agent constructs its own space representation, M. We define M to be **correct** with respect to (A,Υ) if

$$KB \wedge A \wedge \Delta(M) \models \Upsilon$$

Note that A and Υ are expressed in the language of KB while M is expresed in the agent's language.

Shanahan's space formalization only asks for the locations and shapes of objects. In Shanahan's work, space is considered a real-valued coordinate system[6], shape is represented as a collection of straight lines, and the set of agent's actions is $\{$go,stop,rotate(r)$\}$. *KB* is based on these choices to represent space, shape and actions. It does not specify what the behavior of the agent should be: it specifies what space information can be derived from the actual behaviour and observations of the agent.

Planning can be thought of as the inverse operation to temporal projection, and temporal projection in the event calculus is naturally cast as a deductive task. Consequently, planning in the event calculus can be considered as an abductive task. Given a domain description Σ, a conjunction Γ of goals, and a description of the initial situation, Δ_N, a *plan* is a consistent conjuction Δ_p of *Happens* and a temporal ordering formulae such that

$$\Sigma \wedge \Delta_N \wedge \Delta_p \models \Gamma.$$

This logical characterization of the event calculus planning is analgous to Green's logical characterization of situational calculus planning [Green, 1969].

10.3 Qualitative Representation of Space

Qualnav model [Kuipers and Levitt, 1988, Levitt and Lawton, 1990]. The Qualnav model provides a computable theory that integrates qualitative, topological representations of large-scale space with quantitative, metric ones. A difference from the TOUR model, Qualnav considers unstructured environments, with significant perceptual events, called landmarks, scattered throughout this two-dimensional space.

Qualitative navigation is carried out by identifiying *landmark-pair boundaries* (LPBs) between landmarks. Roughly speaking, a LPB between landmarks L_1 and L_2 is a virtual line between L_1 and L_2. This line divides the space into two distinct regions. If the agent

[6]He does not rule out the adoption of qualitative approaches to spatial reasoning.

can observe the landmarks, it can decide then which side of this line it is on. LPBs give rise to a topological division of the ground surface into observable regions of localization, *the orientation regions*. Crossing boundaries between orientation regions leads to a qualitative sense of path planning based on perceptual information. As a landmark-recognizing vision system moves through large-scale space, it builds a visual memory of the interlocking sequence of orientation regions it has traversed through. Adjacency of orientation regions in visual memory can be determined by sharing a common but opposite orientation LPB. If two regions have a common boundary, it is possible to move between them by tracking the landmarks as the agent moves towards the boundary. Thus, *visual memory is an undirected graph where nodes are orientation regions, and arcs join adjacent regions*.

Hernandez [Hernandez, 1994] provides a model for the qualitative representation of positional information in 2-D space based on topological and orientation relations. Using Freska's notion of *conceptual neighborhood for qualitative relations* [Freksa, 1992], algorithms for transforming between explicit reference frames and a canonical one, as well as for composing spatial relations are defined. In addition, [Hernandez, 1994] presents a variety of mechanisms to reason with qualitative representations in general, and qualitative representations of 2-D positional information in particular. These mechanisms are based on the idea of modifying domain independent *constraint satisfaction* techniques to account for the particular constraints of the spatial domain. Moreover, special data structures are used for improving the efficiency of the algorithms. In particular, *abstract maps*, are introduced, containing for each object in a scene a data structure with the same neighborhood structure as the domain requires for the task at hand. A change of view, for example, can then easily accomplished diagrammatically by "rotating" the labels of the orientation with respect to the intrinsic one.

Hernandez [Hernandez, 1994] presents an extensive review of the different approaches to the representation of spatial knowledge (see chapter eight). The integration of metric and path knowledge with qualitative representations of space is not covered in this work. As for path knowledge, Hernandez states that

> "the positional information implicitly contained in paths can be gradually assimilated in form of topological and orientation relations by a process in which positional relations are established for the (virtual) places at which directional change occur. The positions of the actual objects in the scene are then derived by constraint propagation".

10.4 Robotics

Two main areas of robotics are primarily related to our work: map building and autonomous navigation. The map building problem refers as to how the robot creates a spatial representation of its enviroment. Two common spatial representations are geometric and topological maps. A geometric map represents objects according to their absolute geometric relationships (more later). In contrast, a topological map records the geometric relationships between the observed features rather than their absolute position with respect to an arbitrary coordinate frame of reference (see 10.1). The problem of robot navigation corresponds to reliably go from one place to another in the environment. During the navigation, the robot uses its sensors to create a map of its local environment. This local map is then compared to a global spatial representation in order to calculate its position in the environment. We are interested in how the spatial representation supports the "sense-plan-act" navigation cycle. Next we review some of the work we have borrowed ideas from. We refer the reader to Kortenkamp et al.'s book [Kortenkamp *et al.*, 1998] for an extensive review of AI mobile robot systems and techniques.

10.4.1 Metrical maps

The most common representation of geometric map data is a *certainty grid* [Elfes, 1987, Borestein and Koren, 1991]. In a certainty grid approach, sensor readings are placed into the grid by using probability profiles that describe the algorithm's certainty about the existence of objects at individual grid cell. When positioning with respect to a metrical map, sensor derived geometric maps must be matched against a global map of a large area. This is often a formidable difficulty because of the robot's position error. Using topological information has proven be useful in order to reduce the effect of position error when constructing a metrical map [Thrun, 1998, Thrun *et al.*, 1998]. We refer the reader to Borestein's book [Borenstein *et al.*, 1996] (Chapter 8) for a review of the problems and advantages using metrical maps as well as a description of several systems using this spatial representation.

10.4.2 Fuzzy control

Konolige et al. [Konolige *et al.*, 1995] propose an approach for integrating planning and control based on *behavior schemas*, which link physical movements to abstract action descriptions. Behavior schemas describe behaviors of an agent, expressed as trajectories of control actions in an environment, and goals can be defined as predicates on these trajectories. The proposed methodology is summarized as follows [Konolige *et al.*, 1995]:

> “Our approach to integrating planning and control has focussed on grounding the ingredients of planning in physical actions, using the tools of multivalued logics. We started from the definition of basic types of movements, or *control schemas*, and of the way they can be combined or blended to form complex behaviors. Then we have “lifted” control schemas to the level of abstract actions in the environment. Here, we have used two key notions: the notion of *embedding* in the environment, by anchoring the agent’s internal state to external objects through perception, and the notion of *context*, or circumstances of execution. Finally, we have linked behaviors to goals, expressed as sets of satisfactory executions. The good behaviors for a goal are those that, when executed in the appropriate context, produce executions that satisfy the goal. And we have proven, under certain hypotheses, that composing behaviors creates a new behavior that is good for the composition of the corresponding goals. This result is the basis for automatic planning of complex behavior, and we have shown how traditional AI techniques for deliberation and means-ends analysis can be readily adapted to generate complex controllers for given goals”.

10.4.3 Probabilistic navigation

In probabilistic navigation, the location of a robot is represented by a probability distribution over the possible locations of the robot. The robot’s position is updated based on the actions it executed as well as observations gathered during navigation. The method takes into account various sources of uncertainty, including approximate knowledge of the environment, and actuator and sensor uncertainty [Simmons and Koening, 1995, Koening and Simmons, 1998, Nourbakhsh, 1998]. A partially observable Markov decision process (POMDP)[7] model is constructed from topological information about the connectivity of the environment, approximate distance information, plus sensor and actuator characteristics.

[7] See section A.1.2, page 203.

Chapter 11

Conclusions and Future Work

What have we done?. We have taken an informal description of the SSH [Kuipers and Byun, 1988, Kuipers and Byun, 1991, Kuipers, 2000] and given a formal account of some aspects of this theory. In addition, we have extended the theory to handle perceptual aliasing, environments with self intersecting and convergent paths, as well as to deal with local metrical information uncertainty. The new description of the SSH is independent of the agent's exploration strategy and the possible implementations of the theory. This in contrast to the previous SSH descriptions as well as applications of topological maps in robotics [Choset and Nagatani, 2001]. Nevertheless, we have taken the new SSH description as a specification for a program able to keep track of different topological maps consistent with the agent's experiences in the environment. This program supports different exploration strategies as well as facilitating map disambiguation when the case arises.

A logical account of the SSH causal, topological and local metrical theories was given using Nested Abnormality theories [Lifschitz, 1995]. The minimality conditions embedded in the formalization defined the preferred models associated with the theories. In chapters 4 through 8 we illustrated the main properties of the new theories. In particular we showed how the minimal models associated with these theories are adequate models for the spatial knowledge an agent has about its environment. We also illustrated how the different levels of the representation assume different spatial properties about both the environment and the actions performed by the agent. These spatial properties play the role of "filters" the agent applies in order to distinguish the different environment states it has visited.

In the process of formalizing the SSH we have improved the theory in the following aspects:

- Extended the SSH ontologies to include distinctive states, explicitly mention actions at the topological level, and deal with local metrical information uncertainty.

- Handled perceptual aliasing.
- Associated a spatial representation with the SSH causal level.
- Defined the models of the SSH causal and topological theories.
- Added a theory of regions.
- Extended the topological theory to handle environments with convergent and self intersecting paths.

Distinctive states have been included as explicit objects in the theory. The predicates ceq and teq (chapters 4 and 5) were introduced to denote when two distinctive states are equal given causal and topological information, respectively. Local metrical information imposes further constraints on distinctive states (chapter 7). Having distinctive states as objects allows the theory to deal with perceptual aliasing (i.e. environment states that look the same). Once perceptual aliasing is introduced, more than one model of the theory could explain a given set of experiences. In such cases, new information could prove environment states to be different although they were previously believed equal, and viceversa. Our theory captures this non-monotonicity property of map building.

At the SSH causal level we introduced a spatial representation, that of the causal graph. A causal graph is a deterministic finite automaton (DFA) where states are ceq equivalence classes. Actions at the causal level convey patterns of experience, but no spatial configuration. At the SSH topological level, qualitative spatial information in terms of *turn* and *travel* actions is explained in terms of places, paths and regions. Local metrical information associated with action execution is explained in terms of distance among places on a path and angle among paths that intersect at a place. While the ontology of the topological and metrical levels is more elaborated than the one for the causal level, it is easier to build this representation than it is to distinguish environment states based only on view-action-view sequences.

We studied the appropriateness as well as the limitations of the theory. In particular, we illustrated how when sensor information is weak or the environment is symmetric, the SSH topological map may not have the same structure as the "real environment". In such cases, the topological map has fewer places and paths than the ones one would expect (see page 24). Nevertheless, the model is sound in that any view-action-view sequence it predicts will indeed be attainable in the environment.

From a pragmatical point of view, we illustrated how different navigation architectures are supported by the SSH (appendix A) and we tested these architectures in simulation and the physical robot Vulcan (chapter 9). Although for map building purposes actions are considered deterministic, during navigation errors in action execution occur. We defined different approaches that the SSH can accommodate in order to deal with errors in action executions and location uncertainty. Similarly, we defined how the SSH representation can be used for planning, in particular, how regions support hierarchical planning and execution.

In summary, we have defined the SSH models associated with a set of experiences. The resulting theory exhibits some of the agent's map building characteristics: non-monotonicity, use of different sources of information, multiple models when using weak sensors or when the environment is symmetric. The theory's implementation supports different robot exploration strategies and navigation strategies. It is our aim, that this theory as well as the algorithms proposed will constitute the standard for map building applications based on topological maps.

11.1 Future work

There are several different areas of future research linked to this work. These areas include: extensions of the theory, dealing with non-structured (open) environments, and reasoning with multiple spatial representations. We discuss these areas in turn.

11.1.1 Extension for the current theory

Our theory assumes static environments. This assumption is embedded in two axioms: actions are deterministic and distinctive states have a unique view. The first axiom relies on the control level satisfying the SSH closure properties. The second axiom relies on the agent having perceptual recognition abilities such that the same view is associated with any environment state "close enough" to a given distinctive state. However, actions could fail (i.e. miss a distinctive state) and environment states could have different views associated with then (e.g. a door being closed or opened). How should the theory be extended to deal with non-static environments?. Should probabilistic modeling be used instead of the current logical approach?.

A major point of the SSH theory is that the topological map can be built even in the presence of no metrical information. Nevertheless, whenever the agent decides to create a region and assign locations to places in that region, path shapes should be taken into account. How should a path's shape be represented and reasoned about?. Are qualitative description of path shape enough?

11.1.2 Learning the control level

While the SSH control level implementation was not the focus of this dissertation, many interesting questions need to be solved to carry on such implementation. First at all, one has to translate sensory input into views. Views are then seen as classes describing sensory input. How does a robot learns these classes? How many classes should it learn?. Second, the agent control laws are predefined. Different representations of the same environment result from using different sets of views and control laws. How does the agent decides what set of control laws are appropriate for exploring the environment?[1]. Third, the control level has to render actions deterministic. While we rely on control theory to satisfy this requirement, learning control law parameters could benefit from machine learning techniques. In particular, as the agent repeatedly explores the environment, one expects a more reliable execution of control laws.

11.1.3 Open environments

We have tested the SSH in office-like environments and simulation. What does it take to have a SSH robot navigating non-structured open environment, say a university campus? or a museum? Will the current theory be useful? How much does one have to engineer the control level to define the "right" set of actions to handle each environment? Certainly being able to learn the control level (as stated above) will greatly simplify the task of exploring these environment. We also anticipate that the use of different topological hierarchies (see 11.1.4) as well as the representation of local space will be important for these tasks.

In order to engineer a SSH robot, local representations of space should be incorporated. For example, it is easier to navigate a cluttered room based on a reactive vision module or an occupancy metrical representation, than it is in terms of control laws moving the robot between distinctive states. Techniques to do this navigation exists, and the SSH incorporates these local space representations, but the question remains of when these changes of representation occur, and how the agent keeps the relations among these representations.

[1]See 11.1.4 below.

11.1.4 Reasoning with multiple spatial representations

The SSH is a hierarchy of coexisting spatial representations. At the same time, the SSH topological level admits a hierarchical representation based on regions. There are other kind of topological hierarchies based on scale granularity or relations among actions. These approaches admit building more detailed maps based on a existing map. For instance, the agent first builds the map using a general control law (like *follow-corridor*) which is then refined when executing a particular instance of such control law (like *follow-corridor-stopping-at-doorways*). From the relation among the actions, one could relate both representations. The question remains of how/why an agent builds such hierarchies, how they are related, and how/when an agent uses them for spatial reasoning?

Appendix A

Using the SSH

The appropriateness of the SSH representation derives from how well it supports large scale navigation. Navigation is understood here as the task of getting from one place to another. This process involves three main aspects: localization, planning, and plan execution. Localization is the process of knowing the current location of the agent in the environment. Planning is determining the actions that should be executed to get to the destination place, and plan execution is the process of carrying out these actions.

Different architectures exist for implementing robot navigation. The major goal of these architectures is to show how the agent reaches its destination place despite errors in action execution as well as noise in sensory input. They vary on how the cycle of localization, planning and plan execution is carried out, as well as on the kind of information the spatial representation should provide for their success. In this chapter we show how the SSH is suitable as the spatial representation for different navigation architectures. **Our goal is to show how the spatial representation (the SSH) is used by different existing navigation architectures rather than to define "the" SSH navigation architecture**.

This chapter is organized as follows. Section A.1 illustrates how uncertainty about the agent's location can be represented in the context of the SSH. In section A.2 we show how SSH topological paths and regions are used when planning a route to a goal location. Different navigation architectures supported by the SSH are presented in section A.3.

A.1 Location

The *location* of the agent in the environment is described in terms of the *region*, *place*, and the *path* the agent is at. **We assume that a map of the environment is given to the agent**.

The agent may not know its actual location w.r.t. this map as actions may fail or views are not enough to discern the location. Consequently, the agent has to represent the uncertainty associated with its current location and act according to it. Next we will present two different representations of uncertainty that can be accommodated when using the SSH: in the first one, a set of locations represents the possible locations the agent is at; in the second representation, a probability distribution is kept over the universe of locations. In both cases, one has to provide a *state transition function* defining the effect of actions when executed at particular states (locations), as well as to provide an *observation model* describing the effect of making a particular observation at a particular state.

A.1.1 Representing location knowledge

Suppose the agent's current location is represented by a set of locations (states), Σ, the agent believes it may be at. Under this representation, given an action a, one has to specify the effect of a on Σ, which we will denote by Σ_a. By definition,

$$\Sigma_a \ = \ \{ s' : \ \exists s \in \Sigma \ , \ \langle \, s, a, s' \, \rangle \} \tag{A.1}$$

where $\langle \, s, a, s' \, \rangle$ denotes the fact that state s' is a possible result of executing action a on state s. The tuples $\langle \, s', a, s' \, \rangle$ define the possible state transitions associated with the agent actions. Different languages can be used to specify these transitions. For example, STRIPS ([Fikes and Nilsson, 1971]), A_C ([Baral and Gelfond, 1997]), *C* ([Giunchiglia and Lifschitz, 1998]).[1]

Definition A.1 is akin to the one given by Bacchus, Halpern and Levesque in the context of reasoning about knowledge in the situation calculus ([Bacchus *et al.*, 1999], page 182). They in turn draw in Moore's work [Moore, 1979, Moore, 1985] on using *possible world semantics* to formally reason about knowledge and actions. Here we have adapted their definition in the context of transition based approaches to reason about actions ([Gelfond and Lifschitz, 1998, Son and Baral, 2001, Lobo *et al.*, 1997]). Example 35 (page 200) illustrates how definition A.1 works. In particular, we will show how as the agent executes actions and makes observations, the set Σ captures the agent's *knowledge* about its actual location. A formal study of how the knowledge of the agent changes as it performs actions is outside the scope of this work. The reader is referred to [Moore, 1979, Moore, 1985, Scherl and Levesque, 1993, Bacchus *et al.*, 1999, Son and Baral, 2001] for formal accounts of how to reason about actions and knowledge.

[1]We will use *C* to illustrate the concepts presented in this section.

We use the language *C* ([Giunchiglia and Lifschitz, 1998]) to describe the effect of actions on the agent's location. The fluent `loc(p,pa,dir)` is used to represent that the agent is at *place p* facing direction *dir* of *path pa*.[2] Constraints between the different elements describing a location can be derived from the SSH topological axioms (see chapter 5). For example, if the agent is at place p on path pa, it should be the case that p belongs to path pa (i.e. $on(pa,p)$ is true). The topological map is described in terms of the following predicates (see chapters 5,6,8):

- `on(pa,p)` : place *p* is on path *pa*.
- `order(pa,dir,p,q)`: place *q* is after place *p* on direction dir of path *pa*.
- `nextPlace(pa,dir,p,q)`: place *q* is the next place after place *p* on path *pa*.
- `viewsAt(p,v)`: *v* is a view occurring at place *p*.
- `totheRightOf(p,pa,dir,pa1,dir1), totheLeftOf(p,pa,dir,pa1,dir1)` : if the agent is place p facing on direction dir of path pa, after executing a turn right (left) action, the agent will be facing on direction $dir1$ of path $pa1$.

Views at the SSH causal level (chapter 4) are associated with places and used as evidence for the agent to be at a given place. We use the predicate `viewsAt(p,v)` to represent that view *v* occurs at place *p*. Whether a view occurs at a place can be determined from the distinctive states associated with the place. Formally,

$$viewsAt(p,v) \equiv \exists ds \; \{at(ds,p) \wedge view(ds,v)\} \quad .$$

In addition to moving in the environment the agent can *sense* the environment and determine the view at its current location. The transitions needed by equation A.1 are described by the following theory:[3]

Notice that we assume actions to be deterministic during map building, and explicitly model action errors during navigation. Think of this as somebody giving you a map of a city, and you using it for navigation. While visiting the city, you might get lost or miss an intersection. You use the map as a reference to deduce where you may be at given that these errors occurred. *Errors in travel actions* are modeled by considering a travel action as a non-deterministic action, whose effect is to move the agent along the same path but to a

[2]In this presentation we do not include regions in the agent's location.

[3]The theory is expressed in the language C [Giunchiglia and Lifschitz, 1998] augmented with sorts as recognized by Ccalc [McCain, 1999].

```
:- sorts paths; path_dir ; places; views ; action >> (sensingAction ; genericAction).
:- variables
   DIR, DIR1 :: path_dir;
   GA, GA1 :: genericAction; SA, SA  :: sensingAction;
   V :: views;  PA, PA1 :: paths; P, Q    :: places.
:- constants
   pos, neg :: path_dir ;
   loc(places,paths,path_dir) :: inertialFluent;
   travel, turnRight, turnLeft, turnAround :: genericAction;
   observe_view, observe(views) :: sensingAction.

% locations have to be consistent with the map
% The agent cannot be at two different locations at the same time
always loc(P,PA,DIR) ->> on(PA,P).
caused -loc(Q,PA1,DIR1) if loc(P,PA,DIR) & -(P=Q && PA=PA1 && DIR=DIR1).

% Effect of travel and turn
travel    may cause loc(Q,PA,DIR) if loc(P,PA,DIR) & placeAfter(PA,P,DIR,Q).
turnRight causes loc(P,PA1,DIR1) if loc(P,PA,DIR) & nextRight(P,PA,DIR,PA1,DIR1).
turnLeft  causes loc(P,PA1,DIR1) if loc(P,PA,DIR) & nextLeft(P,PA,DIR,PA1,DIR1).
turnAround  causes loc(P,PA,neg) if loc(P,PA,pos).
turnAround  causes loc(P,PA,pos) if loc(P,PA,neg).

% Action preconditions.  The symbols \/Q... means exists Q ...
nonexecutable travel if loc(P,PA,DIR) & -(\/Q:placeAfter(PA,DIR,P,Q)).
nonexecutable turnRight if loc(P,PA,DIR) & -(\/PA1:nextRight(P,PA,DIR,PA1,DIR1)).
nonexecutable turnLeft if loc(P,PA,DIR) & -(\/PA1:nextLeft(P,PA,DIR,PA1,DIR1)).

% sensing actions.
nonexecutable observe(V) if loc(P,PA,DIR) & -viewsAt(P,V).
nonexecutable observe_view if -(\/V:o(observe(V))).
nonexecutable observe(V) if -o(observe_view).

% do not allow concurrent generic actions
nonexecutable GA & GA1 if (GA @< GA1).

% D. not allow concurrent sensing and generic actions. Recall that sensing actions
% (see observe_view) aremodeled by concurrent actions.
nonexecutable SA & GA.
```

Figure A.1: Theory describing a transition model associated with equation A.1. Errors when traveling are modeled by non-deterministic actions. Sensing actions are modeled by concurrent actions not affecting any fluent in the theory and indicating the conditions under which they are executable.

place after the place in which the action is performed.[4] *Sensing actions* (i.e. $observe\_view$, $observe(V)$) are modeled by not affecting any fluent in the theory and by indicating the conditions under which they are executable (e.g. it is possible to observe a view V in the current place only if V is associated with the place). We model the action of sensing a view, $observe\_view$, as a *concurrent* action: $observe\_view$ is executed if and only if at least one of the actions $observe(V)$ is executed.[5] The next example illustrates how these ideas work.

Example 35

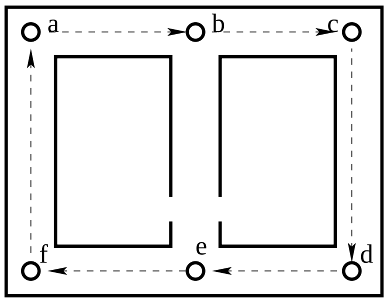


Figure A.2: . The agent visits places $a, b, \ldots, f, a$ in the order suggested by the arrows. Corners *a* and *d* look alike to the agent. Corners *c* and *f* look alike to the agent. Places *b* and *e* have unique views that differentiate them from the other places.

Consider the environment depicted in figure A.2, whose topological map is described by the following Prolog program:[6]

```
viewsAt(a,va).                      viewsAt(b,vb).                      viewsAt(c,vc).
viewsAt(d,va).                      viewsAt(e,ve).                      viewsAt(f,vc).
nextPlace(pa,pos,a,b).              nextPlace(pa,pos,b,c).              nextPlace(pa1,pos,c,d).
nextPlace(pa2,pos,d,e).             nextPlace(pa2,pos,e,f).             nextPlace(pa3,pos,f,a).
totheRightOf(c,pa,pos,pa1,pos).     totheRightOf(d,pa1,pos,pa2,pos).
totheRightOf(f,pa2,pos,pa3,pos).
nextRight(c,pa,pos,pa1,pos).        nextRight(d,pa1,pos,pa2,pos).
nextRight(f,pa2,pos,pa3,pos).       nextRight(a,pa3,pa).

    nextPlace(PA,neg,P,Q) :-                nextPlace(PA,pos,Q,P).
    totheLeftOf(P,PA,DIR,PA1,DIR1) :-       totheRightOf(P,PA1,DIR1,PA,DIR).
    order(PA,DIR,P,Q) :-                    nextPlace(PA,DIR,P,Q).
    order(PA,DIR,P,Q) :-                    nextPlace(PA,DIR,P,R), order(PA,DIR,R,Q).
    on(PA,P) :-                             nextPlace(PA,P,Q) ; nextPlace(PA,Q,P).
```

[4]See "effect of travel" clause in the program.

[5]As a result of performing $observe\_view$ the agent gets a view V. We model this fact by saying that the agent performs concurrently the set of actions $\{observe\_view,\ observe(V)\}$. See "sensing actions" clauses in the program.

[6]The consequences of this program correspond to the topological map we are describing.

According to our theory on page 199, part of the transition diagram associated with this map is depicted on figure A.3.[7]

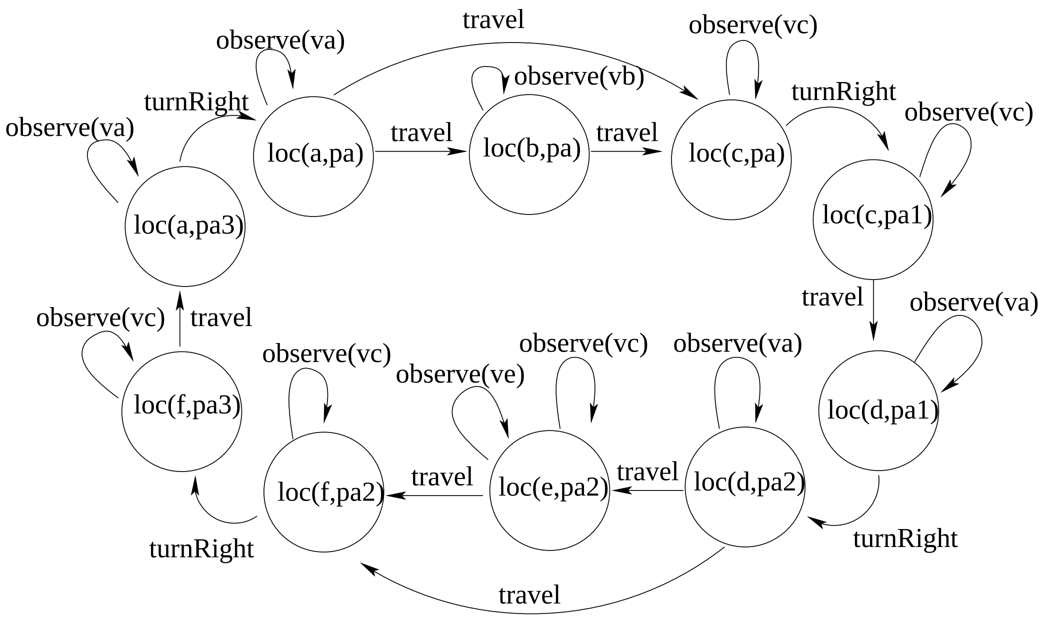


Figure A.3: Transition diagram associated with the environment in figure A.2.

Notice that transitions labeled by sensing actions (i.e. $observe(V)$) leave the system in the same state. Moreover, no transition labeled *observe(V)* exists from a state whose place does not have *V* associated with it. Sensing actions are knowledge producing actions as illustrated next. Suppose the current agent location belief is the set

$$\Sigma \;=\; \{loc(a, pa, pos),\; loc(b, pa, pos)\}\;\; .$$

Suppose the agent makes an observation and obtains view va. Then, according to definition A.1 and the transition diagram in figure A.3,

$$\Sigma_{\{observe\_view, observe(va)\}} \;=\; \{loc(a, pa, pos)\}\quad .$$

Having observed *va*, the agent has ruled out *loc(b,pa,pos)* as a possible location, and it now knows that it is at *loc(a,pa,pos)*.

[7]For readability we have omitted the complete transition diagram. For instance, some transitions are not represented in the diagram (transitions associated with $turnLeft$ and $turnAround$ actions). Only locations along the positive direction of paths are represented (i.e. $loc(p, pa)$ should be read $loc(p, pa, pos)$ in the diagram). Finally, transitions labeled by $observe(V)$ actions should be labeled by the set $\{observe\_view,\; observe(V)\}$.

The execution of actions other than sensing actions may provide information to the agent. Consider the following example. Suppose the agent initially does not know its location, and performs the sequence of actions

$$travel\ ; travel\ \ .$$

The agent's initial location belief is represented by the set

$$\{loc(a, pa3, pos)\ ,\ loc(a, pa, pos)\ ,\ loc(b, pa, pos)\ ,\ loc(c, pa, pos)\ ,\ loc(c, pa1, pos)\ ,\ loc(d, pa1, pos)\ ,\ loc(d, pa2, pos)\ ,\ loc(e, pa2, pos)\ ,\ loc(f, pa2, pos)\ ,\ loc(f, pa3, pos)\}$$

After performing the first travel action, the new location belief becomes

$$\{loc(a, pa3, pos)\ ,\ loc(b, pa, pos)\ ,\ loc(c, pa, pos)\ ,\ loc(d, pa1, pos)\ loc(e, pa2, pos)\ ,\ loc(f, pa2, pos)\}\ \ .$$

Finally, after the second travel action, the current location belief becomes

$$\{loc(c, pa, pos)\ ,\ loc(f, pa2, pos)\}\ \ .$$

Either the agent is at *loc(c,pa,pos)* or at *loc(f,pa2,pos)*. Notice that the execution of travel actions has restricted the possible locations the agent might be at. The agent will have to execute the actions

$$turnRight\ ; travel\ ; turnRight\ ; travel\ ; observe\_view$$

to disambiguate its location.

"Reasoning" back on time is done by considering the history of location believes $\Sigma_0, \Sigma_1, \ldots, \Sigma_n$, where, $\Sigma_i = \Sigma_{i-1,a_i}$,[8] and the operator Σ_a^{-1} defined as follows:

$$\Sigma_a^{-1}\ =\ \{s :\ \exists s' \in \Sigma\ \langle s, a, s' \rangle\}\ \ . \qquad \text{(A.2)}$$

One then considers the sequence $\Sigma'_0, \Sigma'_1, \ldots, \Sigma'_n$, where $\Sigma'_n\ =\ \Sigma_n$ and $\Sigma'_{i-1}\ =\ \Sigma_{i-1} \cap (\Sigma'_i)^{-1}_{a_i}$. For example, from $a_0 = travel$, $a_1 = travel$,

$$\Sigma_2\ =\ \{loc(c, pa, pos)\ ,\ loc(f, pa2, pos)\}\ \ ,$$

and

$$\Sigma_1\ =\ \{loc(a, pa3, pos)\ ,\ loc(b, pa, pos)\ ,\ loc(c, pa, pos)\ loc(d, pa1, pos)\ ,\ loc(e, pa2, pos)\ ,\ loc(f, pa2, pos)\}$$

[8]The initial belief Σ_0 is given.

as above, we conclude

$$\Sigma_1' = \{loc(b, pa, pos)\, ,\, loc(e, pa2, pos)\} \quad .$$

By a further application of equation A.2 the agent concludes that its initial location before executing $travel; travel$ was either *loc(a,pa,pos)* or *loc(d,pa2,pos)*.
$\{$*end of example*$\}$

A.1.2 Probabilistic representations of location

Under the probabilistic approach to representing location uncertainty, a probability distribution is kept over the universe of locations. Transition and observation models have to be provided defining the effect of actions on the different states. Based on these models, a rule has to be defined on how to update the probability distribution after an action is executed. Next we illustrate the use of POMDP[9] in the context of the SSH. The reader is referred to [Shafer, 1976, Duboi and Prade, 1988, Bacchus *et al.*, 1999] for other possible approaches.

A POMDP is a tuple (S,A,T,Θ,O) [10] where: **S** is a finite set of states, **A** is a finite set of actions, **T** is a state transition model of the environment, which is a function mapping elements from SxA into discrete probability distributions over S, **Θ** is a finite set of possible observations, and **O** is an observation function mapping AxS into discrete probability distributions over Θ.

We write $P(s'|a,s)$ for *the probability that the environment will make a transition from state s to state s' when action a is executed*. By definition $P(s'|a,s) = T(s,a,s')$. We write $P(o|a,s)$ for *the probability of making observation o after having executed action a and ended in state s*. By definition, $P(o|a,s) = O(a,s,o)$.

According to Bayes rule, given a current distribution P_{prior}, an action a, and an observation o, we can calculate the distribution after performing a and observing o , $P_{posterior}$, as follows:

$$P_{posterior}(s') = Pr(s'|a,o,P_{prior}) \tag{A.3}$$

[9]POMDP stands for Partially Observable Markov Decision Process.

[10]A POMDP includes also a reward function mapping SxA to the real numbers that specify the instantaneous reward that the agent derives from taking an action in a state (see [Puterman, 1994]).

$$
\begin{aligned}
&= \frac{Pr(o|s', a, P_{prior})Pr(s'|a, P_{prior})}{Pr(o|a, P_{prior})} \\
&= \frac{P(o|a, s')\sum_{s \in S} P(s'|a, s)P_{prior}(s)}{Pr(o|a, P_{prior})}
\end{aligned}
$$

where $Pr(o|a, P_{prior})$ is a normalizing factor defined as

$$Pr(o|a, P_{prior}) = \sum_{s' \in S} P(o|a, s') \sum_{s \in S} P(s'|a, s)P_{prior}(s) \ .$$

Example 36

Consider the same environment as in example 35 (figure A.2, page 200). The POMDP associated with this environment is such that its states are the set of possible locations, actions are SSH actions (i.e. travel, turnRight, turnLeft), observations are the SSH views, and the transition and observation models are derived from the transition diagram illustrated in figure A.3 augmented with probabilities. The actual values of these probabilities have to be learned by the agent as it explores the environment (see [Simmons and Koening, 1995, Basye *et al.*, 1995, Koenig and Simmons, 1996, Engelson and McDermott, 1992b]). In this example we have assigned these probabilities by assuming that $Pr(V|\, loc(p, pa)) = Pr(V|\, loc(p', pa'))$ for all p and p' such that $viewAt(p, V) \equiv viewAt(p', V)$.[11] As for the transition model, we have assumed that after traveling it is more probable to be in the next place in the current path that in any other place on the path. Figure A.4 illustrates the corresponding transition model.

Suppose the agent does not know its current location. This fact is represented by a uniform distribution over the set of possible locations. After performing a $travel$ action, the new distribution is:

```
Pr(loc(a,pa))  = 0,     Pr(loc(a,pa3)) = 1/6,
Pr(loc(b,pa))  = 2/15,  Pr(loc(c,pa))  = 1/5,
Pr(loc(c,pa1)) = 0,     Pr(loc(d,pa1)) = 1/6,
Pr(loc(d,pa2)) = 0,     Pr(loc(e,pa2)) = 2/15,
Pr(loc(f,pa2)) = 1/5,   Pr(loc(f,pa3)) = 0.
```

Notice that the agent's most probable states are either $loc(c, pa)$ or $loc(f, pa2)$. Suppose now that the agent observes view vc. The new distribution becomes:

[11]To simplify the example we only consider the positive direction of paths in a location, and write $loc(p, pa)$ instead of $loc(p, pa, pos)$. Moreover, we assume that the observation function depends only on the state and not in the executed action that brought the agent to this state.

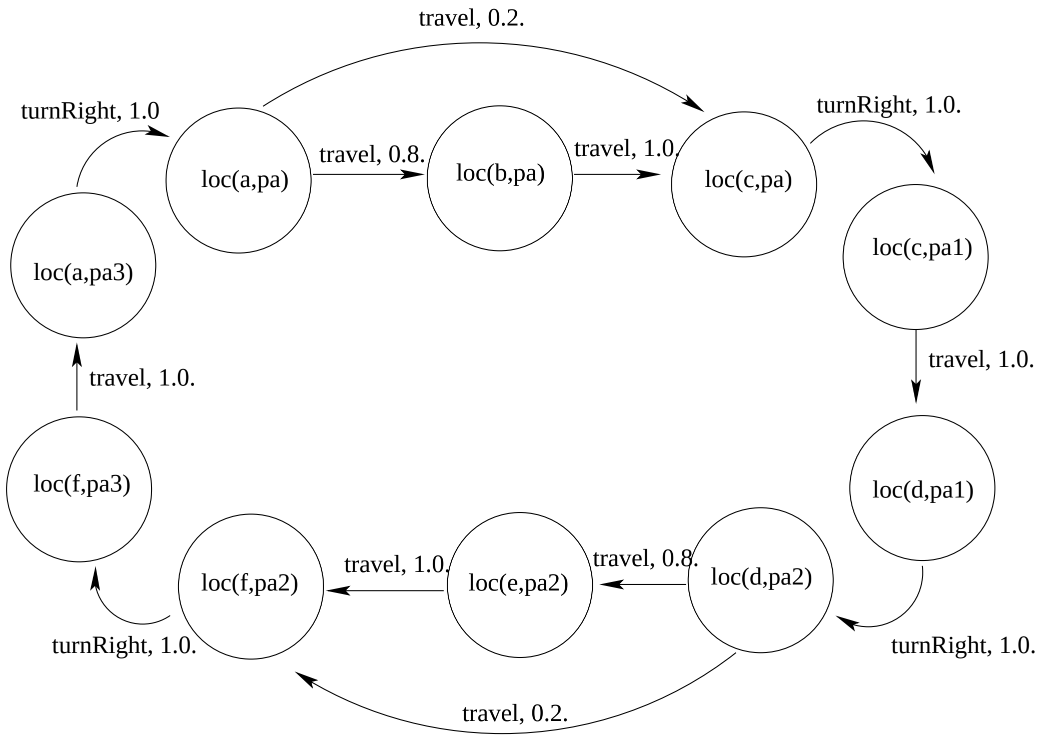


Figure A.4: POMDP transition model associated with the environment in figure A.2.

```
Pr(loc(a,pa))  = 0,     Pr(loc(a,pa3)) = 0,
Pr(loc(b,pa))  = 0,     Pr(loc(c,pa))  = 1/2,
Pr(loc(c,pa1)) = 0,     Pr(loc(d,pa1)) = 0,
Pr(loc(d,pa2)) = 0,     Pr(loc(e,pa2)) = 0,
Pr(loc(f,pa2)) = 1/2,   Pr(loc(f,pa3)) = 0.
```

{*end of example*}

In summary, the SSH representation can accommodate different representations of the agent's location. The SSH topological map was used to define state transition models used during navigation. In these models, action's errors were accounted for. Views at the causal level were used to provide evidence of whether the agent is at a given place. Different theories to reason about the effect of actions and observations in the agent's location were discussed in association with these models.

A.2 Planning

The problem of finding a sequence of actions that takes the robot from place *p* to place *q* can be reduced to a graph search problem. This does not come as a surprise given the graph-like structure of the SSH topological map. Paths and regions at the SSH topological level are used to expedite the search for a route to a goal destination. Next we illustrate how tailored planners can be defined to efficiently use paths and regions for route finding. Generic planners can also be used with the SSH (section A.2.3).

A.2.1 Using topological paths for planning

Two basic search graphs can be associated with the SSH topological map. The first one is defined as follows:

- The nodes of the graph are the places of the topological map.
- An edge with label *pa,dir* from place *p* to place *q* exists whenever $order(pa, dir, p, q)$ is the case.

The second one is a bipartite graph such that:

- Nodes correspond to the union of places and paths in the topological map.
- An edge exists from place p to path pa whenever $on(pa, p)$ is the case.

Notice that a path in the graphs above does not completely specify the sequence of actions to get to the goal. However, enough information exists in the SSH to derive such actions. A path on the graphs above can be easily converted to a route description, or used as a plan skeleton during navigation.

Example 37

Consider finding a plan to go from place *a* to place *d* in the environment of figure A.5. A shortest path between these two places will be

$$(a, c, pa) \ ; \ (c, d, pa1) \ \ .$$

This path can be interpreted as the plan *follow path pa until place c, then take path pa1 and follow it until place d.* Each of the edges in this path represents a sequence of actions: for example, the labeled edge (a, c, pa) represents the sequence of actions $travel \ ; \ travel$. Turn actions need to be included in the plan when a change of path is required. For example, at place *c*, to turn from path *pa* to path *pa1*, the action *turnRight* should be executed. {*end of example*}

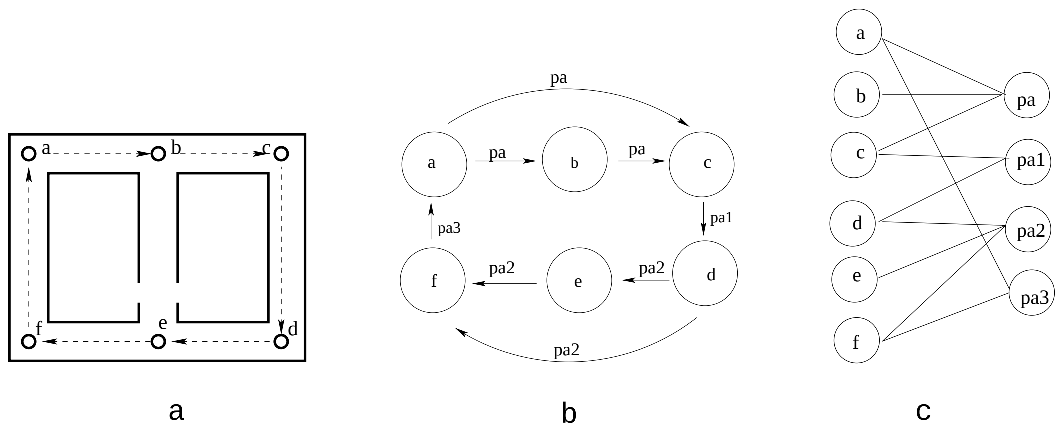


Figure A.5: (b) Search graph associated with the environment in a. (c) Bipartite search graph associated with the environment in a.

A.2.2 Using regions for planning

In chapter 8 we introduced regions in the context of the SSH. Regions were defined to be sets of places with a containment hierarchy among them. Connectivity relations between regions are derived from connectivity relations between places in the regions. This requirement allows us to define a hierarchical route planner as defined in chapter 8 (page 120).

Example 38

Consider the same scenario as in our previous example (page 206). Suppose we have the regions *R1, R2,* and *R* such that $R = \{R1, R2\}$, $R1 = \{a, b, c\}$ and $R2 = \{d, e, f\}$

In order to find a path from place *a* to place *e* the hierarchical planner finds first a route from region *R1* to region *R2*. This path specifies that the edge $(c, d, pa1)$ will be part of the plan.[12] Recursively, the planner finds a route in $R1$[13] from *a* to *c* as well as a path in $R2$ from *d* to *e*. The paths (a, b, pa); (b, c, pa) and $(d, e, pa2)$ are returned respectively and a complete path will then be derived. Note that the route found may be sub-optimal as in this case.
{*end of example*}

[12] A nondeterministic choice will be done in case different edges connect *R1* and *R2*.
[13] A path whose places belong to *R1*.

A.2.3 Conventional planners

While the two sections above show how tailored planners can be used with the SSH, it is possible to use more generic planners. For example, a modification of the causal theory presented in page 199 can be used as input for Ccalc [McCain, 1999] in order to do planning. When planning, we do not model action errors as part of the theory nor do we include sensing actions.[14] We replace the rule

$$loc(Q, PA, DIR)\ caused\ after\ travel\ \&\ loc(P, PA, DIR)\ \&\ order(PA, DIR, P, Q)$$

by

$$loc(Q, PA, DIR)\ caused\ after\ travel\ \&\ loc(P, PA, DIR)\ \&\ nextPlace(PA, DIR, P, Q)$$

in our causal theory. Action errors will be detected and handled by the navigation system, as described in section A.3.

A.3 Navigation

A navigation architecture defines how the agent integrates sensing, planning, and acting within a single system. Such an architecture has to deal with key issues such as uncertainty (in both sensing and action), reliability, and real-time response ([Kortenkamp *et al.*, 1998]). The SSH can be used as the spatial representation for different navigation architectures. Next we outline how the SSH can be used by *observe-plan-act* architectures as well as for architectures based on hierarchical planning and execution. The reader is referred to ([Kortenkamp *et al.*, 1998]) for the state of the art of working mobile robot architectures.

Observe-plan-act architectures

Most navigation architectures follow a `observe-plan-act` loop. They vary on how often each step in the loop is run as well as the actual order and implementation of the steps in the loop. The *observe* step of the loop helps to determine the agent's location as well as to detect failures in the execution of actions. The *plan* step determines what the next action (actions) to execute should be. The *act* step corresponds to carrying out the action (actions) determined in the plan step. Depending on the representation chosen for the agent's location, different schemas are used to implement the *observe-plan* steps.

Under the probabilistic representation of the agent's location (section A.1.2) an action is chosen by assuming that the agent is at the most probable location. A deterministic

[14]The agent will sense the environment as part of its navigation strategy. Our causal theory in page 199 defines how the output of these sensing actions affects the knowledge state of the agent.

plan is then generated from that location to the goal destination. The first action of such plan is executed. Observations are made and the agent location is updated according to equation A.3. The loop *observe-plan-act* continues until the most probable location is the destination goal.

The description above assumes that the probability distribution has a “clear” most probable state. Whenever this is not the case, the agent could change its goal from going to a given destination to gathering relevant information that allows it to localize better. Once the agent is better localized, navigation continues to the destination goal ([Simmons and Koening, 1995, Koening and Simmons, 1998]).

When the agent’s location is represented by a set of possible locations (section A.1.1), the next action to execute is found by selecting l from the set of possible locations, and generating a deterministic plan from l to the destination. The first action of this plan is executed, observations are made and the agent’s location is updated according to equation A.1. In order to select the new current location the agent uses equation A.1 applied to the set $\{l\}$. If the result is the empty set, a new location is selected at random, otherwise a location from this set is chosen as the current location and the loop *observe-plan-act* continues until the set of possible locations is a subset of the set of locations satisfiying the goal. The reader is referred to [Nourbakhsh and Genesereth, 1996, Nourbakhsh, 1998, Baral and Gelfond, 1999] for details on how to implement these ideas.

Hierarchical planning and execution

Hierarchical planning as described in section A.2.2 can be interleaved with plan execution to efficiently accommodate errors in actions and sensing as well as incomplete information. Instead of generating a detailed plan, the *hierarchical planner* represents a plan by a tree whose nodes get expanded as the plan is executed. This tree is traversed in a depth first fashion so that actions for the robot can be generated as soon as possible.

Each node of the plan tree represents a plan step that the *planner* **tells** the *sequencer* to execute. The sequencer either executes this plan step and notifies the planner it did so or **asks** the planner for more information on how to carry on the step. On receiving a notification from the sequencer, the planner either tells the agent to execute the next step in the plan (if the sequencer successfully executed the previous step) or search for a new plan to achieve the goal. On being asked for more information about a plan step, the planner expands the plan step and tells the sequencer to execute a new (probably more refined) step in the plan. The net result of the `tell/ask` interaction between the planner and the sequencer is that we have hierarchical planning and execution. The example below illustrates these ideas.

Consider the scenario in which a robot is turned on and asked to go to the chairman's office (see figure A.6). First notice that the agent does not necessarily know where it is.[15] Consequently, a sensible plan will include actions to decide where it is (possibly by asking the driver).

Suppose the robot decides that it is at the lab (either by asking or by using the last location it was at) as its current location. Once a location has been associated with "the chairman's office", a plan can be derived from the topological map depicted in figure A.6.

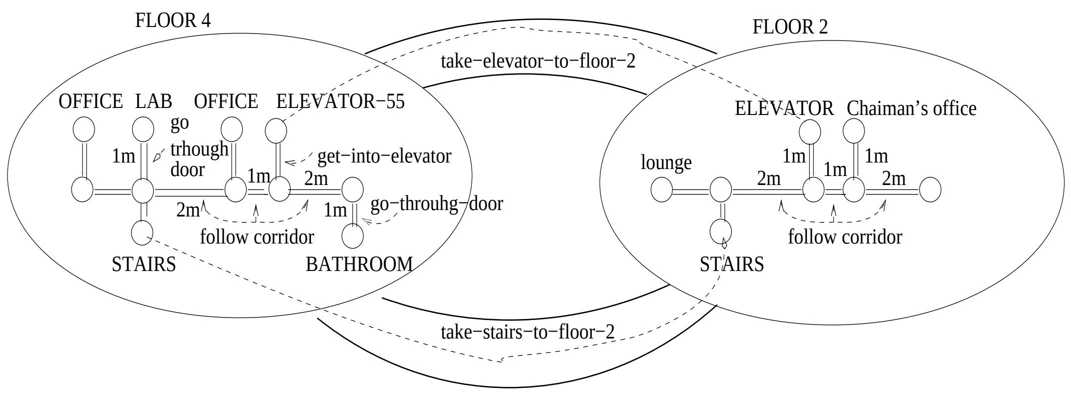


Figure A.6: Going to the chairman's office. Floors are represented as regions in the SSH topological map. These regions are connected by `(take-elevator-to-floor ?x)` actions.

The general plan will be

```
go-to(elevator-55)
take-elevator-to(floor-2)
go-to(chairman-office)
```

Notice that the last step of the plan is the same as the original goal. However, by the time the robot executes this step, the robot will be at floor 2 and consequently closer to the chairman's office. The planner now proceeds in a depth first fashion to expand the plan above. A new plan (restricted to floor 4) will be then generated for the goal `go-to(elevator-55)`. In this case the planner uses the map associated with `floor-4` to solve this problem and finds the plan:

[15]It's safe to assume that it remembers the last place it was at before being shutdown!

```
go-through-door
follow-corridor for about 2m
follow-corridor for about 1m
```

Whether the robot should go left or right at the corridor after going through the door is not explicitly mentioned in the plan. That information will be made explicit during the plan execution. In order to execute the plan above, the planner tells the sequencer to execute each plan step as follows:[16]

```
(define-sequencer-task (step-go-through-door)
  (if (succeed (go-through-door))
    (notify-planner success)
    (ask-planner expand-step))
)
```

The `(notify-planner success)` call allows the planner to monitor the execution of the plan. As steps are executed, the robot's location in the map is updated. On receiving a notification of step execution success, the planner will invoke the execution of the next step in the plan (by generating a plan step similar to the one above).

When the planner is asked to *expand* a plan step, it uses a more detailed map to carry out the step. If this is not possible, a failure in the plan is detected. Failures are taken into account by indicating that either a place or path in the topological map cannot be used for future plans.[17] For example, if the plan step `(go-through-door)` fails, then the robot should conclude that it is still at the lab, and that the path connecting the lab with the corridor should not be used for future plans.

The action `go-through-door` looks like

```
(define-sequencer-task  (go-through-door)
  (if (look-and-found? ?door)
     ((get-close-to ?door) (pass-through ?door))
     (return failure)
  )
)
```

[16]For the purpose of the examples we made up a language to specify the tasks the sequencer should carry on. Special purpose languages exist to specify these tasks: RAPS [Firby, 1994], ESL [Gat, 1996]. See [Kortenkamp *et al.*, 1998].

[17]This implies that the planner has a local memory to indicate changes in the representation.

The action `(look-and-found?  ?door)` will use the sensory system to look for the door. The result of executing this action will be to *anchor* the door so that a reactive procedure can be performed to get to the door. The action `(get-close-to ?door)` will make the robot navigate towards the door. Notice that this procedure can be complicated and might involve planning or using a local map.

If the door is not found, the sequencer will ask the planner how to get to the door. The planner finds a plan by using a map of the lab (plus knowing where in the lab the robot is) as depicted in figure A.7. The resulting plan will be `{follow-wall until at(door-at-lab); get-`

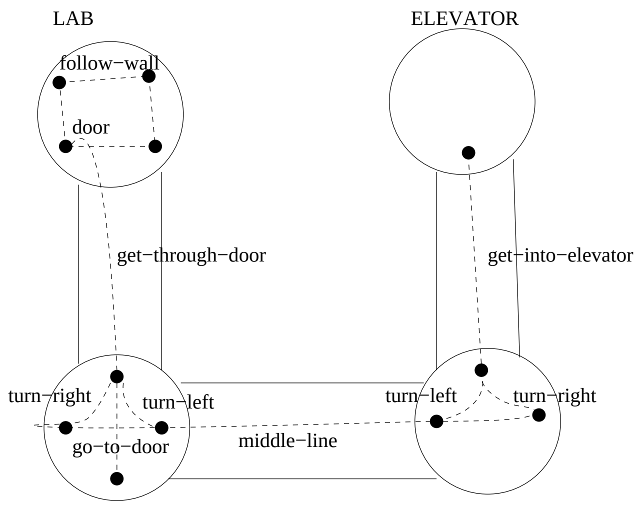


Figure A.7: Detailed maps for the lab, the corridor and the elevator.

A.4 Summary

In this chapter we have shown how the SSH is a suitable spatial representation for navigation. For each of the sub problems associated with navigation, localization, planning, and plan execution, we discussed different approaches supported by the SSH representation.

In order to take into account errors while navigating the environment (e.g. missing an intersection), the SSH supports two different approaches to represent location uncertainty: representing disjunctive location knowledge and probabilistic representations of location. In the former, location is represented by a set of possible states. In the latter, location is represented by a distribution over possible states. In both cases, we defined the transition and observation models associated with the corresponding approaches.

In order to solve planning problems (i.e. finding a route to a goal destination), the SSH supports the use of standard planners as well as special purpose planners exploiting paths and regions at the SSH topological level. We defined the corresponding algorithms for this later case.

Finally, we discussed different navigation architectures supported by the SSH. In addition to observe-plan-act architectures, we defined an architecture combining hierarchical planning and execution. In this architecture, instead of generating a detailed plan, a hierarchical planner represents a plan by a tree whose nodes get expanded as the plan is executed. This tree is traversed in a depth first fashion so that actions for the robot can be generated as soon as possible. We illustrated how this architecture can accomodate recursive plans (i.e. in order to go home, leave the office and go home) as well as recover from errors in action execution.

The different methods described in this chapter illustrate how the SSH supports navigation. In addition, the methods illustrate how the different SSH levels are combined during localization, planning and plan-execution.

Appendix B

Topological Level Properties

In this appendix we prove some properties of the SSH topological theory. Recall the SSH topological theory is defined as follows:

$$
\begin{aligned}
& TT(E) \; = \\
& \quad \mathit{there\ exist\ infinitely\ many\ places}\ , \\
& \quad \mathit{there\ exist\ infinitely\ many\ paths}\ , \\
& \quad \neg\exists p\,[tplace(p) \wedge is\_region(p)]\,, \\
& \quad \neg\exists pa\,[tpath(pa) \wedge route(pa)]\ , \\
& \quad Axioms\ \ 5.1 - 5.2\ , \\
& \quad COMPLETION(E)\ , \\
& \quad Axioms\ 4.1 - 4.11\ , \\
& \quad \langle ds, a, ds'\rangle \wedge \langle ds, a, ds''\rangle \rightarrow ds' = ds'', \qquad (Axiom\ 4.22) \\
& \quad T\_block\ , \\
& \quad \{\ min\ is\_region,\ min\ route\ : \\
& \quad\ \ AT\_block = \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(B.1)} \\
& \quad\ \ \{\ max\ teq : \\
& \quad\quad\ \ \Gamma \\
& \quad\ \ \ \mathbf{circ}\ \ tpath \succ tplace\ \ \mathbf{var}\ \ S\vec{SH}pred \qquad\qquad\qquad\qquad \text{(B.2)} \\
& \quad\ \ \} \\
& \quad \}
\end{aligned}
$$

where Γ is the set of axioms defined in page 41, and $S\vec{SH}pred$ stands for the tuple of predicates $\langle \mathbf{at}, \mathbf{along}, \mathbf{order}, \mathbf{on}, \mathbf{teq}, turn\_eq, travel\_eq, \rangle$.

The properties of the SSH topological theory proved in this appendix are listed below. Let M be a model for $TT(E)$, then

- $M \models \forall pa,\ [tpath(pa) \equiv \exists ds, dir\ along(ds, pa, dir)]$.
- $M \models \forall p,\ [tplace(p) \equiv \exists ds\ at(ds, p)]$.
- The topological map associated with a finite set of experiences E has a finite number of topological paths and a finite number of topological places.
- Any two models of the SSH topological theory have the same number of topological paths and the same number of topological places.

Theorem 8 *Let M be a model of $TT(E)$. Then,*

- $M \models \forall pa,\ [tpath(pa) \equiv \exists ds, dir\ along(ds, pa, dir)]$.
- $M \models \forall p,\ [tplace(p) \equiv \exists ds\ at(ds, p)]$.

Proof.

$$\begin{array}{l} CIRC[\Gamma; tpath \succ tplace; SSHpred] \\ \equiv \qquad \{proposition\ 15\ in\ [Lifschitz, 1994]\} \\ CIRC[\Gamma; tpath; tplace, SSHpred] \wedge CIRC[\Gamma; tpath, tplace; SSHpred] \\ \rightarrow \qquad \{def.\ of\ circumscription\} \\ CIRC[\Gamma; tpath] \end{array}$$

Since $\Gamma = \Gamma'(tpath) \wedge [along(ds, pa, dir) \rightarrow tpath(pa)]$ where $\Gamma'(tpath)$ is negative, then

$$\begin{array}{l} CIRC[\Gamma; tpath] \\ \equiv \\ CIRC[\Gamma'(tpath) \wedge [along(ds, pa, dir) \rightarrow tpath(pa)]; tpath] \\ \equiv \qquad \{proposition\ 4\ in\ [Lifschitz, 1994]\} \\ \Gamma'(tpath) \wedge CIRC[along(ds, pa, dir) \rightarrow tpath(pa); tpath] \\ \rightarrow \qquad \{proposition\ 1\ in\ [Lifschitz, 1994]\} \\ [\exists ds, dir\ along(ds, pa, dir)] \equiv tpath(pa) \end{array}$$

Similarly, $\Gamma = \Gamma' \wedge [at(ds, p) \rightarrow tplace(p)]$ where $tpath$ does not occur in Γ'. Then,

$$CIRC[\Gamma; tpath \succ tplace; SSHpred]$$

$$
\begin{array}{l}
\rightarrow \qquad \{see\ above\} \\
CIRC[\Gamma; tpath, tplace; SSHpred] \\
\rightarrow \qquad \{def.\ parallel\ circumscription\} \\
CIRC[\Gamma; tpath, tplace] \\
\rightarrow \qquad \{def.\ parallel\ circumscription\} \\
CIRC[\Gamma' \wedge [at(ds,p) \rightarrow tplace(p)]; tplace] \\
\equiv \qquad \{propositions\ 1\ and\ 4\ in\ [Lifschitz, 1994]\} \\
\Gamma' \wedge [\exists ds, at(ds,p)] \equiv tplace(p)
\end{array}
$$

□

Theorem 4. The topological map associated with a finite set of experiences E has a finite number of topological paths and a finite number of topological places.

Proof. Since a distinctive state is along at most one topological path (axiom 5.24, page 42), theorem 8 implies that for any model M of $TT(E)$ there is an injection from $tpath^M$ into $distinctive\text{-}states^M$. Since $distinctive\text{-}states^M$ is finite so is $tpath^M$.

Similarly, since distinctive states are at a unique topological place (axiom 5.16, page 41), from theorem 8 we conclude that the set of topological places in a model of $TT(E)$ is finite. □

Theorem 5. Any two models of the SSH topological theory have the same number of topological paths and the same number of topological places.

Proof. In order to prove that two models M_1 and M_2 of $TT(E)$ have the same number of topological paths (tpaths) and the same number of topological places (tplaces), it is enough to show that this is the case for models of the AT_block (block B.1). Suppose that $tpath^{M_1}$ has less elements than $tpath^{M_2}$, and so there exists an injection $\phi : tpath^{M_1} \rightarrow tpath^{M_2}$. *One can extend ϕ to define an isomorphism from M_1 into M_2', such that $M_2' \prec M_2$, where $\prec$ is the order defined by the circumscription policy B.2.*[1] In fact,

- Let $\phi : tplace^{M_1} \rightarrow places^{M_2}$ be an injection. Such an injection exists since $tplace^{M_1}$ is finite and $places^{M_2}$ is infinite.
- Let $\phi : S^{M_1} \rightarrow S^{M_2}$ be the identity over the sorts (S)of distinctive states, actions,

[1]This proves that M_1 and M_2 have the same number of topological paths.

views, schemas, path types and path directions. Recall we assumed a *Herbrand* interpretation for these sorts, where the corresponding universes are defined by the constant symbols in E.

The function ϕ above defines an isomorphic embedding from M_1 into M_2 in the standard way. In fact, $\phi(M_1) = M_2'$ is defined as follows:

- $tpath^{M_2'} = \phi(tpath^{M_1}), tplace^{M_2'} = \phi(tplace^{M_1})$.
- $teq^{M_2'} = \phi(teq^{M_1}) = \{teq(ds_1, ds_2) \ : \ M_1 \models teq(ds_1, ds_2)\} \ = \ teq^{M_1}$.
- $at^{M_2'} = \phi(at^{M_1}) = \{at(ds, \phi(p)) \ : \ M_1 \models at(ds, p)\}$.
- $along^{M_2'} = \phi(along^{M_1}) = \{along(ds, \phi(pa), dir) \ : \ M_1 \models along(ds, pa, dir)\}$.
- $order^{M_2'} = \phi(order^{M_1}) = \{order(\phi(pa), dir, \phi(p), \phi(q)) : M_1 \models order(pa, dir, p, q)\}$.
- $on^{M_2'} = \phi(on^{M_1}) = \{on(\phi(pa), \phi(p)) \ : \ M_1 \models on(pa, p)\}$.
- $turn\_eq^{M_2'} = \phi(turn\_eq^{M_1}) = \ turn\_eq^{M1}$.
- $travel\_eq^{M_2'} = \phi(travel\_eq^{M_1}) = \ travel\_eq^{M1}$.

Notice that the language of Γ is defined by $\{tpath, tplace\} \cup SSHpred$. Thus $M_1 \models \Gamma$ implies $\phi(M_1) \models \Gamma$.[2] Since $\phi(tpath^{M_1}) \subset tpath^{M_2}$, then $\phi(M_1) \prec M_2$, and so M_2 is not minimal, and is therefore not a model of $TT(E)$. It follows that M_1 and M_2 have the same number of topological paths.

Similar argument shows that M_1 and M_2 have the same number of topological places. If not, there would exists $\phi : tpath^{M_1} \rightarrow tpath^{M_2}$ a bijection and $\phi : tplace^{M_1} \rightarrow tplace^{M_2}$ an injection that allows us to apply the same argument as above. $\square$

Theorem 6. Let ds_1 be a distinctive state symbol such that

$$\forall ds_2 \notin [ds_1]_{\widehat{turn}}, \ [ds_2]_{teq} \cap [ds_1]_{\widehat{turn}} \ = \ \emptyset \ . \qquad \text{(B.3)}$$

Then

$$\forall ds_2 \notin [ds_1]_{\widehat{turn}}, \ place(ds_2) \neq place(ds_1) \ .$$

[2]Notice that the circumscription policy varies all predicates in the language of Γ, and ϕ is the identity over all constant symbols in the theory, for otherwise, $\phi(M_1) \models \Gamma$ is not necessarily the case. In general the interpretations of a unary predicate (set) under a circumscriptive theory do not have the same number of elements. For example, consider the models of $CIRC[(P(0) \wedge P(1)) \vee P(2); P]$, where the interpretation of P could have one or two elements (this example is due to Vladimir Lifschitz).

Proof. The hypothesis of the theorem implies that

$$\forall ds_2 \notin [ds_1]_{\widehat{turn}},\ \neg turn\_eq(ds_2, ds_1) \ \ .$$

Indeed,

$$\begin{array}{l} turn\_eq(ds_1, ds_2) \\ \equiv \\ \exists b_0, \ldots, b_n, b_{0'}, \ldots, b_{n'} \ \ s.t. \\ \qquad \bullet\ b_0 = ds_2,\ b_{n'} = ds_1\ , \\ \qquad \bullet\ teq(b_i, b_{i'}),\ i = 0, \ldots, n \\ \qquad \bullet\ \widehat{turn}(b_{i'}, b_{i+1}),\ i = 0, \ldots, n-1\ . \end{array}$$

Let $1 \le j \le n$ such that $\left[\forall j \le k \le n,\ b_{k'} \in [ds_1]_{\widehat{turn}}\right]$ and $b_{(j-1)'} \notin [ds_1]_{\widehat{turn}}$. Notice that such a j exists since $ds_1 = b_{0'} \notin [ds_1]_{\widehat{turn}}$ and $ds_1 = b_{n'} \in [ds_1]_{\widehat{turn}}$. Consequently,

$$\begin{array}{ll} turn\_eq(ds_1, ds_2) & \\ \rightarrow & \\ b_{j'} \in [ds_1]_{\widehat{turn}} & \\ \rightarrow & \{teq(b_j, b_{j'})\} \\ [b_j]_{teq} \cap [ds_1]_{\widehat{turn}} \neq \emptyset & \\ \rightarrow & \{B.3\} \\ b_j \in [ds_1]_{\widehat{turn}} & \\ \rightarrow & \{\widehat{turn}(b_{(j-1)'}, b_j)\} \\ b_{(j-1)'} \in [ds_1]_{\widehat{turn}} & \\ \rightarrow & \\ false & \end{array}$$

Thus $\neg turn\_eq(ds_2, ds_1)$ should be the case. □

Appendix C

Nested Abnormality theories

In this appendix we define circumscription and nested abnormalities theories following [Lifschitz, 1994, Lifschitz, 1995].

The main idea of circumscription is to consider, instead of arbitrary models of an axiom set, only the models that satisfy a certain minimality condition (usually set inclusion). Mathematically, circumscription is defined as a syntactic transformation of logical formulas. It transforms a sentence A into a stronger sentence A^*, such that the models of A^* are precisely the minimal models of A.

Definition 12 (Circumscription)

Let $A(P, Z_1, \ldots, Z_m)$ be a sentence containing a predicate constant P and object, function and/or predicate constants $Z_1, \ldots, Z_m$ (and possibly other object, function and predicate constants). The *circumscription of P in A with varied* $Z_1, \ldots, Z_m$ is the sentence

$$A(P, Z_1, \ldots, Z_m) \wedge \neg \exists p, z_1, \ldots, z_m \, [A(p, z_1, \ldots, z_m) \wedge p < P] \tag{C.1}$$

where $p < P$ denotes the formula

$$\forall x \ \{p(x) \rightarrow P(x)\} \ \wedge \exists x \{\neg p(x) \wedge P(x)\} \quad .$$

We denote formula C.1 by $CIRC\,[A; P; Z]$.
$\{$*end of definition*$\}$

Intuitively, the models of $CIRC\,[A; P; Z]$ are the models of A in which the extent of P cannot be smaller without losing the property A, even at the price of changing the interpretations of the constants Z. In order to make this claim precisely, the following order, $\leq^{P;Z}$, is defined among structures of the language of A.

Definition 13 ($\leq^{P;Z}$)

Let M_1 and M_2 be two structures for a given one-sorted language. Then $M_1 \leq^{P;Z} M_2$ whenever

- $|M_1| = |M_2|$,[1]
- $M_1[C] = M_2[C]$,[2] for every constant C which is different from P and does not belong to Z,
- $M_1[P] \subseteq M_2[P]$

$\{$*end of definition*$\}$

In other words, $M_1 \leq^{P;Z} M_2$ means that M_1 and M_2 differ only in how they interpret P and Z, and the extent of P in M_1 is a subset of its extent in M_2. Proposition 1 in [Lifschitz, 1994] states that

Theorem 9 *A structure M is a model of $CIRC[A;P;Z]$ if and only if M is minimal relative to $\leq^{P;Z}$.*

Example 39

Circumscription is usually used in order to formalize default associated with an axiomatic theory. Suppose the we would like to represent the default "Normally a block in on the table". Suppose block B_1 is not on the table and let B_2 denotes another block. The circumscriptive theory below allow us to conclude $Ontable(B_2)$:

$$Block(x) \wedge \neg Ab(x) \rightarrow Ontable(x) \tag{C.2}$$

$$\neg Ontable(B_1) \tag{C.3}$$

$$Block(B_1),\ Block(B_2),\ \ B_1 \neq B_2 \tag{C.4}$$

$$\textbf{circ}\ \ Ab\ \ \textbf{var}\ \ Ontable \tag{C.5}$$

where we have extended our notation such that the above should be understood as

$$CIRC[C.2 \wedge C.3 \wedge C.4; Ab; Ontable] \quad .$$

[1]For a structure M, $|M|$ denotes the universe of M.
[2]$M[C]$ denotes the interpretation of C in M.

In the above theory, it is the case that $Ab(x) \equiv x = B_1$, and consequently $Ontable(B_2)$ follows. The role of the predicate Ab is to single out the blocks that are *"abnormal"* relative to the default (C.2).
{*end of example*}

It is often convenient to arrange different defaults by assigning priorities to them. For example, consider formalizing the enhancement of the theory above in which "Blocks are usually heavy", and "heavy block are usually not in the table". Next we define two extensions to the basic definition of circumscription: parallel and prioritized circumscription.

Definition 14 (Parallel Circumscription)

The *parallel circumscription*

$$CIRC\left[A; P^1, \ldots, P^n; Z\right]$$

is the sentence

$$A(P, Z) \wedge \neg\exists p, z\left[A(p, z) \wedge p \prec P\right] \quad ,$$

where P stands for the tuple of predicates $P^1, \ldots, P^n$ and $p \prec P$ stands for the formula $\forall\, 1 \leq i \leq n\; p^i \leq P^i \;\; \wedge \exists\; 1 \leq i \leq n\; p^i < P^i$
{*end of definition*}

The parallel circumscription of several predicates has a simple model theoretics characterization, similar to the one presented by theorem 9. When P is a tuple $P^1, \ldots, P^n$, the relation $M_1 \leq^{P;Z} M_2$ between structures M_1 and M_2 is defined as before, except that the condition $M_1\,[P] \subseteq M_2\,[P]$ is replaced by $M_1\,[P^i] \subseteq M_2\,[P^i]$ for all $i = 1, \ldots, n$.

Definition 15 (Prioritized Circumscription)

The *prioritized circumscription*

$$CIRC\left[A; P^1 \succ \ldots \succ P^n; Z\right]$$

is the sentence

$$A(P, Z) \wedge \neg\exists p, z\left[A(p, z) \wedge p \prec P\right] \quad ,$$

where P stands for the tuple of predicates $P^1, \ldots, P^n$ and $p \prec P$ stands for the formula

$$\bigvee_{i=1}^{n}\left(\bigwedge_{j=1}^{i-1}(p^j = P^j) \wedge (p^i < P^i)\right) \quad .$$

$\{$*end of definition*$\}$

The formula $p \prec P$ defines a *lexicographic* order among the predicates in p and P. When $k = 1$ it becomes $p < P$; if $k = 2$, it becomes

$$(p^1 < P^1) \vee ((p^1 = P^1) \rightarrow (p^2 < P^2)) \ \ .$$

Proposition 15 in [Lifschitz, 1994] shows that prioritized circumscription can be reduced to parallel circumscription as follows:

Theorem 10 *The circumscription* $CIRC\,[A; P^1 \succ \ldots \succ P^n; Z]$ *is equivalent to*

$$\bigwedge_{i=1}^{n} CIRC\left[A; P^i; P^{i+1}, \ldots, P^n, Z\right] \ \ .$$

Notation 2

$CIRC\,[A; P^1 \succ \ldots \neg P_i \ldots \succ P^n; Z]$ stands for the formula

$$CIRC\left[A \wedge not\_P_i \equiv \neg P_i; P^1 \succ \ldots not\_P_i \ldots \succ P^n; Z, P_i\right]$$

where not_P_i is a new constant predicate not occurring in A. $\{$*end of notation*$\}$

C.1 Nested Abnormality theories (NAT's)

Nested abnormality theories allows one to apply the circumscription operator to a subset of axioms, by structuring the knowledge base (the theory) into blocks. Each block can be viewed as a group of axioms that describes a certain collection of predicates and functions, and the nesting of blocks reflects the dependence of these descriptions on each other.

Definition 16 (NAT's)

Consider a second-order language L that does *not* include Ab among its symbols. For every natural number k, by L_k we denote the language obtained from L by adding Ab as a k-ary predicate constant. *Blocks* are defined recursively as follows: For any k and any list of function and/or predicate constants $C_1, \ldots, C_m$ of L, if each of $A_1, \ldots, A_n$ is a formula of L_k or a *block*, then $\{C_1, \ldots, C_m \ : \ A_1, \ldots, A_n\}$ is a *block*. The last expression reads: $C_1, \ldots, C_m$ are such that $A_1, \ldots, A_n$. About $C_1, \ldots, C_m$ we say that they are *described* by this block.

The semantics of NAT's is characterized by a map φ that translates blocks into sentences of L. It is convenient to make φ defined also on formulas of the languages L_k. If A is such a formula, then $\varphi(A)$ stands for the universal closure of A. For blocks we define, recursively:

$$\varphi\{C_1,\ldots,C_m \ : \ A_1,\ldots,A_n\} = \exists ab\, CIRC\,[\varphi A_1,\ldots,\varphi A_n \ : \ ab \ : \ C_1,\ldots,C_m] \ .$$

{*end of definition*}

Example 40

Consider the standard example: objects normally don't fly: birds normally do; canaries are birds; Tweety is a canary. These assertions can be formalized as the NAT whose only axiom is

$$\begin{array}{l}
\{Flies \ : \\
\quad Flies(x) \rightarrow Ab(x), \\
\quad \{\, Flies \ : \\
\quad\ \ Bird(x) \wedge \neg Ab(x) \rightarrow Flies(x), \\
\quad\ \ Canary(x) \rightarrow Bird(x), \\
\quad\ \ Canary(Tweety) \\
\quad \} \\
\}
\end{array}$$

The outer block describe the ability of objects to fly; the inner block gives more specific information about the ability of *birds* to fly. Each occurrence of the predicate Ab is "local" to its block, and so, the two occurrences of the predicate Ab refer two "unrelated" predicates though we use the same name.
{*end of example*}

Most often, it is desirable not to mention the predicate Ab at all. We will adopt the following notations:

- $\{C_1,\ldots,C_m, min\ P \ : \ A_1,\ldots,A_n\}$ stands for

$$\{C_1,\ldots,C_m, P \ : \ P(x) \rightarrow Ab(x),\ A_1,\ldots,A_n\}$$

- $\{C_1, \ldots, C_m, max\ P\ :\ A_1, \ldots, A_n\}$ stands for

$$\{C_1, \ldots, C_m, P\ :\ \neg Ab(x) \rightarrow P(x),\ A_1, \ldots, A_n\}$$

Using this notation, we could rewrite our previous example as

$$
\begin{array}{l}
\{min\ Flies\ : \\
\quad \{\ Flies\ : \\
\quad\ \ Bird(x) \wedge \neg Ab(x) \rightarrow Flies(x), \\
\quad\ \ Canary(x) \rightarrow Bird(x), \\
\quad\ \ Canary(Tweety) \\
\quad \} \\
\}
\end{array}
$$

where we dispense the occurrence of one Ab predicate. The reader is referred to [McCarthy, 1980, McCarthy, 1986, Lifschitz, 1994, Lifschitz, 1995] for a complete survey of the uses and properties of circumscriptions and NATs.

Definition 17

We extend the definition of *blocks* as follows: if A is a block, so is $CIRC[A; P^1 \succ \ldots \succ P^n; Z]$. The semantics of NATs is extended such that

$$\phi CIRC[A; P^1 \succ \ldots \succ P^n; Z] = CIRC[\phi A; P^1 \succ \ldots \succ P^n; Z]\ .$$

$\{$*end of notation*$\}$

As the next theorem shows, in some cases prioritized circumscription can be expressed using NAT's. In these cases however, the notation for prioritized circumscription is more compact than its equivalent NAT's. This motivates our previous definition.

Theorem 11 *Let A be a sentence such that Ab does not occur in A. Then,*

$$CIRC[A; P \succ Q; Z] = \{Z,\ min\ Q:\ \{Z,\ Q,\ min\ P:\ A\}\}\ .$$

Proof.

$\{Z,\ min\, Q:\ \{Z,\ Q,\ min\, P:\ A\}\}$

$\equiv \quad Proposition\ 1\ in[Lifschitz, 1995]$

$\{Z,\ min\, Q:\ CIRC[A;P;Z,Q]\}$

$\equiv \quad Proposition\ 1\ in[Lifschitz, 1995]$

$CIRC[CIRC[A;P;Z,Q];Q;Z]$

$\equiv \quad CIRC's\ definition$

$CIRC[A;P;Z,Q] \wedge \neg\exists q,z\{A(P,q,z) \wedge \neg\exists p,q',z'[A(p,q',z') \wedge p < P] \wedge q < Q\}$

$\equiv \quad logic$

$CIRC[A;P;Z,Q] \wedge [\neg\exists p,q',z'[A(p,q',z') \wedge p < P] \rightarrow \neg\exists q,z\{A(P,q,z) \wedge q < Q\}]$

$\equiv \quad CIRC's\ definition\ and\ logic$

$CIRC[A;P;Z,Q] \wedge \neg\exists q,z\{A(P,q,z) \wedge q < Q\}$

$\equiv \quad CIRC's\ definition\ and\ logic;$

$CIRC[A;P;Z,Q] \wedge CIRC[A;Q,Z]$

$\equiv \quad Proposition\ 10$

$CIRC[A;P \succ Q;Z]$

$\square$

Appendix D

Answer Sets

In this appendix we defined the answer set semantics for a logic program as defined in [Lifschitz, 1999, Gelfond and Lifschitz, 1991].

Consider a set of propositional symbols, called **atoms**. A **literal** is an expression of the form A or $\neg A$, where A is an atom (we call the symbol $\neg$ "classical negation", to distinguish it from the symbol *not* used for negation as failure). A rule element is an expression of the form L or *not* L, where L is a literal. A **rule** is an ordered pair

$$Head \leftarrow Body \tag{D.1}$$

where $Head$ and $Body$ are finite sets of rule elements. If

$$Head = \{L_1, \ldots, L_k, not\, L_{k+1}, \ldots, not\, L_l\}$$

and

$$Body = \{L_{l+1}, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n\}$$

($n \geq m \geq l \geq k \geq 0$) then we write (D.1) as

$$L_1; \ldots, L_k; not\, L_{k+1}, \ldots, not\, L_l \leftarrow L_{l+1}, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n \quad .$$

A rule (D.1) is a **constraint** if $Head = \emptyset$. A program is a set of rules.

The notion of an answer set is defined first for program that do not contain negation as failure ($l = k$ and $n = m$ in every rule of the program). Let Π be such program, and let X be a consistent set of literals. We say that X is **closed** under Π if, for every rule in Π, $Head \cap X \neq \emptyset$ whenever $Body \subseteq X$. We say that X is an **answer set** of Π if X is minimal among the sets closed under Π (relative to set inclusion).

Example 41

The program

$$p; q \leftarrow$$
$$\neg r \leftarrow p$$

has two answer sets: $\{p, \neg r\}$ and $\{q\}$. If we add the constraint

$$\leftarrow q$$

to this program, we will get a program whose only answer set is $\{p, \neg r\}$ (see theorem 12 below). $\{$*end of example*$\}$

To extend the definition of an answer set to programs with negation as failure, take any program Π, and let X be a consistent set of literals. The **reduct** Π^X of Π relative to X is the set of rules

$$L_1; \dots, L_k; \leftarrow L_{l+1}, \dots, L_m \quad ,$$

for all rules (D.1) in Π such that X contains all the literals $L_{k+1}, \dots, L_l$ but does not contain any of the $L_{m+1}, \dots, L_n$. Thus Π^X is a program without negation as failure. We say that X is an **answer set** for Π if X is an answer set for Π^X.

Example 42

The program

$$p \leftarrow not\, q$$
$$q \leftarrow not\, p$$

has two answer sets: $\{p\}$ and $\{q\}$. $\{$*end of example*$\}$

Adding a constraint to a program affects its collection of answer sets by eliminating the answer sets that "violate" this constraint. Next we prove this property of answer sets.

Theorem 12 *Let Π_1 and Π_2 be logic programs such that Π_2 is obtained from Π_1 by adding a set of constraints C (i.e. $\Pi_2 = \Pi_1 \cup C$). Let X be a consistent set of literals. Then X is an answer set for Π_2 if and only if X is an answer set for Π_1 such that for each rule $\leftarrow L_1, \dots, L_m, not\, L_{m+1}, \dots, not\, L_n \;\; \in C$, $\{L_1, \dots, L_m\} \not\subseteq X$ whenever X does not contain any of $L_{m+1}, \dots, L_n$.*

Proof. Let X and Y be consistent sets of literals. Let $C^X(Y)$ denote the fact that Y does not violates any constraint in C^X, that is, if $\leftarrow L_1, \ldots, L_m \in C^X$ then $\{L_1, \ldots, L_m\} \not\subseteq Y$. In particular,

$$Y \subseteq X \wedge C^X(X) \rightarrow C^X(Y) \tag{D.2}$$

Moreover, it is the case that

$$C^X(Y) \equiv Y \textit{ is closed under } C^X \tag{D.3}$$

In fact, if Y is closed under C^X and $\leftarrow L_1, \ldots, L_m \in C^X$, then $\{L_1, \ldots, L_m\} \not\subseteq Y$ for otherwise $Y \cap \emptyset \neq \emptyset$. Conversely, if $C^X(Y)$ is the case, it is easy to see that Y is closed under C^X. From the definition of reduct we have that $\Pi_2^X = \Pi_1^X \cup C^X$. Using (D.2) it is then the case that

$$\{Y : \mathit{closed}_{\Pi_2^X}(Y)\} = \{Y : \mathit{closed}_{\Pi_1^X}(Y) \wedge C^X(Y)\} \tag{D.4}$$

From facts (D.2) and (D.4) the theorem follows:

$$\begin{aligned}
& X \textit{ answer set for } \Pi_2 \\
\equiv \quad & X \textit{ answer set for } \Pi_2^X \\
\equiv \quad & X \textit{ minimal of } \{Y : \mathit{closed}_{\Pi_2^X}(Y)\} \\
\overset{(D.4)}{\equiv} \quad & X \textit{ minimal of } \{Y : \mathit{closed}_{\Pi_1^X}(Y) \wedge C^X(Y)\} \\
\equiv \quad & C^X(X) \wedge \forall Y \left(Y \neq X \wedge Y \subseteq X \wedge C^X(Y) \rightarrow \neg \mathit{closed}_{\Pi_1^X}(Y)\right) \\
\overset{(D.2)}{\equiv} \quad & C^X(X) \wedge \forall Y \left(Y \neq X \wedge Y \subseteq X \rightarrow \neg \mathit{closed}_{\Pi_1^X}(Y)\right) \\
\equiv \quad & C^X(X) \wedge X \textit{ answer set for } \Pi_1^X \\
\equiv \quad & C^X(X) \wedge X \textit{ answer set for } \Pi_1
\end{aligned}$$

□

Appendix E

Ceq Properties

In this appendix we provide proofs for the different properties of the predicated ceq defined in chapter 4 (Page 26). We start by proving that predicate ceq is indeed an equivalence relation.[1] We then illustrate that it in general it is necessary to explicitly ask ceq to be symmetric and transitive.[2] We show that have the agent completely explored the environment, then the maximization principle defining ceq will guaranty that ceq is an equivalence relation without explicitly requiring so (Page 232). Moreover, we show that in this case the predicate ceq captures the idea that two distinctive states are the same if they render the same views under any sequence of actions.

Theorem 1. The predicate ceq is an equivalence relation.
Proof. For the purpose of the proof next we reproduce the definition of the theory *CT(E)*:

$$
\begin{array}{l}
CT(E) \ = \\
\quad COMPLETION(E)\ , \\
\quad Axioms\ 4.1 - 4.11\ , \\
\quad \langle ds, a, ds' \rangle \wedge \langle ds, a, ds'' \rangle \rightarrow ds' = ds'', \qquad (Axiom\ 4.22) \\
\quad CEQ\_block \ = \\
\quad \{\ max\ \ ceq : \\
\quad\ \ ceq(ds, ds') \rightarrow ceq(ds', ds), \\
\quad\ \ ceq(ds, ds') \wedge ceq(ds', ds'') \rightarrow ceq(ds, ds''), \\
\\
\quad\ \ ceq(ds, ds') \rightarrow View(ds, v) \equiv View(ds', v), \\
\quad\ \ ceq(ds_1, ds_2) \wedge \langle ds_1, a, ds'_1 \rangle \wedge \langle ds_2, a, ds'_2 \rangle \rightarrow ceq(ds'_1, ds'_2)
\end{array}
$$

[1] See ceq's definition below.

[2] The maximization associated with ceq's definition does not guaranty these properties.

}

We need to prove that $ceq(ds, ds)$ is the case. Let M_1 be a model for the axioms inside the *CEQ_block* as well as the other axioms of $CT(E)$. Let M_2 be a structure identical to M_1 except that

$$ceq^{M_2}(ds, ds') \equiv ceq^{M_1}(ds, ds') \vee ds = ds' \quad .$$

Our theorem follows once we prove that M_2 is a model for the axioms inside the *CEQ_block*.[3] Next we show why this is the case:

- $M_2 \models ceq(ds, ds') \rightarrow ceq(ds', ds)$. In fact,

$$\begin{array}{ll} \multicolumn{2}{l}{ceq^{M_2}(ds, ds')} \\ \equiv & ceq^{M_1}(ds, ds') \vee ds = ds' \\ \rightarrow & ceq^{M_1}(ds', ds) \vee ds' = ds \\ \equiv & ceq^{M_2}(ds', ds) \end{array}$$

- $M_2 \models ceq(ds, ds') \wedge ceq(ds', ds'') \rightarrow ceq(ds, ds'')$. In fact,

$$\begin{array}{ll} \multicolumn{2}{l}{ceq^{M_2}(ds, ds') \wedge ceq^{M_2}(ds', ds'')} \\ \equiv & \left(ceq^{M_1}(ds, ds') \vee ds = ds'\right) \wedge \left(ceq^{M_1}(ds', ds'') \vee ds' = ds''\right) \\ \equiv & \left(ceq^{M_1}(ds, ds') \wedge ceq^{M_1}(ds', ds'')\right) \vee \left(ds = ds' \wedge ceq^{M_1}(ds', ds'')\right) \vee \\ & \left(ceq^{M_1}(ds, ds') \wedge ds' = ds''\right) \vee \left(ds = ds' \wedge ds' = ds''\right) \\ \rightarrow & ceq^{M_1}(ds, ds'') \vee \left(ds = ds' \wedge ds' = ds''\right) \\ \equiv & ceq^{M_2}(ds, ds'') \end{array}$$

- $M_2 \models ceq(ds, ds') \rightarrow View(ds, v) \equiv View(ds', v)$. In fact,

$$\begin{array}{ll} \multicolumn{2}{l}{ceq^{M_2}(ds, ds')} \\ \equiv & ceq^{M_1}(ds, ds') \vee ds = ds' \\ \rightarrow & \forall v \left[View(ds, v) \equiv View(ds', v)\right] \vee ds = ds' \\ \rightarrow & \forall v \left[View(ds, v) \equiv View(ds', v)\right] \vee \forall v \left[View(ds, v) \equiv View(ds', v)\right] \\ \equiv & View(ds, v) \equiv View(ds', v) \end{array}$$

[3] M_2 satisfies the other axioms in $CT(E)$ since ceq does not occur in them.

- $M_2 \models ceq(ds_1, ds_2) \wedge \langle ds_1, a, ds'_1 \rangle \wedge \langle ds_2, a, ds'_2 \rangle \rightarrow ceq(ds'_1, ds''_2)$. In fact,

$$\begin{array}{ll} & ceq^{M_2}(ds_1, ds_2) \wedge \langle ds_1, a, ds'_1 \rangle \wedge \langle ds_2, a, ds'_2 \rangle \\ \equiv & \Big(ceq^{M_1}(ds_1, ds_2) \wedge \langle ds_1, a, ds'_1 \rangle \wedge \langle ds_2, a, ds'_2 \rangle \Big) \vee \\ & \big(ds_1 = ds_2 \wedge \langle ds_1, a, ds'_1 \rangle \wedge \langle ds_2, a, ds'_2 \rangle \big) \\ \rightarrow & ceq^{M_1}(ds'_1, ds'_2) \vee \big(\langle ds_1, a, ds'_1 \rangle \wedge \langle ds_1, a, ds'_2 \rangle \big) \\ \overset{(4.22)}{\rightarrow} & ceq^{M_1}(ds'_1, ds'_2) \vee ds'_1 = ds'_2 \\ \equiv & ceq^{M_2}(ds'_1, ds'_2) \end{array}$$

□

Axiom 4.22 (i.e. actions are deterministic) is fundamental in the proof above. Without this axiom, we could have a set of experiences like

$$\begin{array}{l} Action\_type(ml, travel)\ , \\ CS(s1, a, ml, b)\ ,\ CS(s2, a, ml, c)\ . \\ View(a, v)\ ,\ View(b, v1)\ ,\ View(c, v2) \end{array}$$

for which $ceq(a, a)$ is not the case.

In general it is not possible to remove the *ceq*'s symmetry and transitivity axioms from inside CEQ_block. Consider the following example.

Example 43

Let E be the set defined by the following formulae:

$$\begin{array}{l} CS(s1, ds_1, a_1, ds_2)\ ,\ CS(s2, ds_1, a_2, ds_3)\ , \\ CS(s3, ds_2, a_3, ds_4)\ ,\ CS(s4, ds_3, a_3, ds_5)\ , \\ View(ds_1, v)\ ,\ View(ds_2, v)\ ,\ View(ds_3, v)\ , \\ View(ds_4, v1)\ ,\ View(ds_5, v2)\ . \end{array}$$

Suppose our definition of CEQ_block were :

$$\begin{array}{l} CEQ\_block\ = \\ \quad \{\ max\ \ ceq : \\ \quad\ ceq(ds, ds') \rightarrow View(ds, v) \equiv View(ds', v), \\ \quad\ ceq(ds_1, ds_2) \wedge \langle ds_1, a, ds'_1 \rangle \wedge \langle ds_2, a, ds'_2 \rangle \rightarrow ceq(ds'_1, ds'_2) \\ \quad \} \end{array}$$

Since $v1 \neq v2$ we conclude $\neg ceq(ds_4, ds_5)$. This in turn implies $\neg ceq(ds_2, ds3)$. However, in order to maximize *ceq* we can make $ceq(ds_1, ds_2) \wedge ceq(ds_1, ds_3) \wedge ceq(ds_2, ds_1) \wedge ceq(ds_3, ds_1) \wedge \forall ds\, ceq(ds, ds)$ to be the case. In such a model, *ceq* is not transitive.
{*end of example*}

There is a special case in which ceq is symmetric and transitive without explicitly tell so. This is the case when the result of every action at every distinctive state is known. In this case, we said that the set of experiences is complete.

Definition 18

A set of experiences E is **complete** whenever

$$E \models \forall a, ds \exists ds' \langle ds, a, ds' \rangle \ \ .$$

{*end of definition*}

Theorem 2. Let E be a complete set of experiences. Let *CT(E)* be defined as follows:

$$\begin{array}{l}
CT(E) \ = \\
\quad COMPLETION(E)\ , \\
\quad Axioms\ 4.1 - 4.11\ , \\
\quad \langle ds, a, ds' \rangle \wedge \langle ds, a, ds'' \rangle \rightarrow ds' = ds'', \qquad (Axiom\ 4.22) \\
\quad CEQ\_block\ = \\
\quad \{\ max\ \ ceq : \\
\quad\ \ ceq(ds, ds') \rightarrow View(ds, v) \equiv View(ds', v), \\
\quad\ \ ceq(ds_1, ds_2) \wedge \langle ds_1, a, ds'_1 \rangle \wedge \langle ds_2, a, ds'_2 \rangle \rightarrow ceq(ds'_1, ds'_2) \\
\quad \}
\end{array}$$

Then the predicate ceq is an equivalence relation.

Proof. Let M_1 be a model for the axioms inside the *CEQ_block* as well as the other axioms of $CT(E)$. We need to prove that it is possible to have a structure M_2 identical to M_1 except that $ceq^{M_1} \subseteq ceq^{M_2}$, M_2 is a model for the axioms inside the *CEQ_block*, the other axioms of $CT(E)$, and ceq^{M_2} is an equivalence class. The proof goes along the lines of theorem's 1 proof. Indeed, the same proof as in theorem 1 allow us to assume that $M1$ is reflexive.

Let M_2 be a model identical to M_1 except that

$$ceq^{M_2}(ds, ds') = ceq^{M_1}(ds, ds') \vee ceq^{M_1}(ds', ds)\ .$$

By definition, ceq^{M_2} is symmetric. We need to prove that M_2 satisfy the axioms inside (the new) CEQ_block:

- $M_2 \models ceq(ds, ds') \rightarrow View(ds, v) \equiv View(ds', v)$. In fact,

$$\begin{aligned}
&ceq^{M_2}(ds, ds') \\
&\quad \equiv \quad ceq^{M_1}(ds, ds') \vee ceq^{M_1}(ds', ds) \\
&\quad \rightarrow \quad \forall v \left[View(ds, v) \equiv View(ds', v)\right] \vee \forall v \left[View(ds', v) \equiv View(ds, v)\right] \\
&\quad \equiv \quad View(ds, v) \equiv View(ds', v)
\end{aligned}$$

- $M_2 \models ceq(ds_1, ds_2) \wedge \langle ds_1, a, ds_1' \rangle \wedge \langle ds_2, a, ds_2' \rangle \rightarrow ceq(ds_1', ds_2')$. In fact,

$$\begin{aligned}
&ceq^{M_2}(ds_1, ds_2) \wedge \langle ds_1, a, ds_1' \rangle \wedge \langle ds_2, a, ds_2' \rangle \\
&\quad \equiv \quad \left[ceq^{M_1}(ds_1, ds_2) \wedge \langle ds_1, a, ds_1' \rangle \wedge \langle ds_2, a, ds_2' \rangle\right] \vee \\
&\quad \quad \left[ceq^{M_1}(ds_2, ds_1) \wedge \langle ds_1, a, ds_1' \rangle \wedge \langle ds_2, a, ds_2' \rangle\right] \\
&\quad \rightarrow \quad ceq^{M_1}(ds_1', ds_2') \vee ceq^{M_1}(ds_2', ds_1') \\
&\quad \equiv \quad ceq^{M_2}(ds_1', ds_2')
\end{aligned}$$

Finally, let M_2 be a model identical to M_1 except that

$$ceq^{M_2} = transitive\_closure(ceq^{M_1}) \; .$$

By definition, ceq^{M_2} is transitive. If ceq^{M_1} is reflexive and symmetric, so is ceq^{M_2}. We need to prove that M_2 satisfies the axioms inside (the new) CEQ_block:

- $M_2 \models ceq(ds, ds') \rightarrow View(ds, v) \equiv View(ds', v)$. In fact,

$$\begin{aligned}
&ceq^{M_2}(ds, ds') \\
&\quad \equiv \quad \exists ds^0, ds^1, \ldots, ds^n \left[ds = ds^0,\, ds' = ds^n,\, ceq^{M_1}(ds^i, ds^{i+1}),\; 0 \le i < n\right] \\
&\quad \rightarrow \quad \exists ds^0, ds^1, \ldots, ds^n \\
&\qquad\qquad \left[ds = ds^0,\, ds' = ds^n,\, View(ds^i, v) \equiv View(ds^{i+1}, v),\; 0 \le i < n\right] \\
&\quad \rightarrow \quad \exists ds^0, ds^n \;\left[ds = ds^0,\, ds' = ds^n, View(ds^0, v) \equiv View(ds^n, v)\right] \\
&\quad \equiv \quad View(ds, v) \equiv View(ds', v)
\end{aligned}$$

- $M_2 \models ceq(ds_1, ds_2) \wedge \langle ds_1, a, ds_1' \rangle \wedge \langle ds_2, a, ds_2' \rangle \rightarrow ceq(ds_1', ds_2')$. In fact,

$$\begin{aligned}
&ceq^{M_2}(ds_1, ds_2) \wedge \langle ds_1, a, ds_1' \rangle \wedge \langle ds_2, a, ds_2' \rangle \\
&\quad \equiv \quad \exists ds^i (1 \le i \le n) \left[ds_1 = ds^1,\, ds_2 = ds^n,\, ceq^{M_1}(ds^i, ds^{i+1}),\; 1 \le i < n\right]
\end{aligned}$$

$$\begin{array}{ll} & \wedge\langle ds_1, a, ds_1'\rangle \wedge \langle ds_2, a, ds_2'\rangle \\ \overset{hyp.}{\rightarrow} & \exists ds^i \exists \langle ds^i, a, ds^{i'}\rangle \\ & \left[ds_1 = ds^1, ds_2 = ds^n, ds_1' = ds^{1'}, ds_2' = ds^{n'}, ceq^{M_1}(ds^i, ds^{i+1}),\ 1 \leq i < n\right] \\ \rightarrow & \exists ds^{i'} \left[ds_1' = ds^{1'},\ ds_2' = ds^{n'},\ ceq^{M_1}(ds^{i'}, ds^{(i+1)'}),\ 1 \leq i < n\right] \\ \equiv & ceq^{M_2}(ds_1', ds_2') \end{array}$$

□

When a set of experiences is complete the predicate *ceq* captures the idea that two distinctive states are the same if they render the same views under any sequence of actions. Assume that E is complete and let $A = a_1, \ldots, a_n$ denote a sequence of actions. The term $A(ds)$ denotes the distinctive state resulting from executing A starting at ds. By definition, $A(ds) = ds$ if $n = 0$, $A(ds) = ds'$ such that $E \models \langle\ \langle a_1, \ldots, a_{n-1}\rangle(ds), a_n, ds'\rangle$. Notice that the definition of $A(ds)$ makes sense since E is complete and actions are deterministic.

Theorem 3. Let E be a complete set of experiences. Then,

$$ceq(ds, ds') \equiv \forall A, v\ \ [View(A(ds), v) \equiv View(A(ds'), v)]\ \ .$$

Proof. Let M_1 be a model for the axioms inside the *CEQ_block* as well as the other axioms of $CT(E)$. Let M_2 be a model identical to M_1 except that

$$ceq^{M_2}(ds, ds') \equiv \forall A, v\ \ [View(A(ds), v) \equiv View(A(ds'), v)]\ \ .$$

By induction in the length of action sequences on can prove that $ceq^{M_1} \subseteq ceq^{M_2}$. Our proof is complete by showing that M_2 satisfies the axioms inside (the new) CEQ_block:

- $M_2 \models ceq(ds, ds') \rightarrow View(ds, v) \equiv View(ds', v)$. In fact, suppose $M_2 \models ceq(ds, ds')$ and consider the empty sequence of actions, $A = \{\}$, $A(ds) = ds$. Then

$$View(ds, V) \equiv View(A(ds), v) \equiv View(A(ds'), v) \equiv View(ds', v)\ .$$

- $M_2 \models ceq(ds_1, ds_2) \wedge \langle ds_1, a, ds_1'\rangle \wedge \langle ds_2, a, ds_2'\rangle \rightarrow ceq(ds_1', ds_2')$. In fact,

$$\begin{array}{ll} & ceq^{M_2}(ds_1', ds_2') \\ \equiv & \forall A, v\ \ [View(A(ds_1'), v) \equiv view(A(ds_2'), v)] \\ \leftarrow & \langle ds_1, a, ds_1'\rangle \wedge \langle ds_2, a, ds_2'\rangle\ \wedge \\ & \quad \forall A, v\ \ [View(aA(ds_1), v) \equiv View(aA(ds_2), v)] \\ \leftarrow & ceq^{M_2}(ds_1, ds_2) \wedge \langle ds_1, a, ds_1'\rangle \wedge \langle ds_2, a, ds_2'\rangle \end{array}$$

□

Appendix F

Logic Program Correctness

Given a set of experiences E, the models of the theory $CT(E)$ (Page 26) indicate under what circumstances it is possible to consider two distinctive states as referring to the same environment state. In order to calculate these models, next we define a logic program such that its answer sets are in a one to one correspondence with the models of $CT(E)$.

Recall that the theory $CT(E)$ is defined as follows:

$$
\begin{aligned}
& CT(E) \; = \\
& \quad COMPLETION(E)\;, && \text{(F.1)} \\
& \quad Axioms\; 4.1 - 4.11\;, \\
& \quad \langle ds, a, ds' \rangle \wedge \langle ds, a, ds'' \rangle \rightarrow ds' = ds'' && \text{(F.2)} \\
& \quad CEQ\_block \; = \\
& \quad \{\; max \;\; ceq : \\
& \quad\; ceq(ds, ds), && \text{(F.3)} \\
& \quad\; ceq(ds, ds') \rightarrow ceq(ds', ds), && \text{(F.4)} \\
& \quad\; ceq(ds, ds') \wedge ceq(ds', ds'') \rightarrow ceq(ds, ds''), && \text{(F.5)} \\
& \\
& \quad\; ceq(ds, ds') \rightarrow View(ds, v) \equiv View(ds', v), && \text{(F.6)} \\
& \quad\; ceq(ds_1, ds_2) \wedge \langle ds_1, a, ds_1' \rangle \wedge \langle ds_2, a, ds_2' \rangle \rightarrow ceq(ds_1', ds_2') && \text{(F.7)} \\
& \quad \}
\end{aligned}
$$

The logic program Π we will consider is defined as follows:

$$
\begin{aligned}
p(X,Y,X,Y) &\leftarrow \; . && \text{(F.8)} \\
p(X,Y,X2,Y1) &\leftarrow \; p(X,Y,X1,Y1), ceq(X1,X2). && \text{(F.9)} \\
p(X,Y,X1,Y2) &\leftarrow \; p(X,Y,X1,Y1), ceq(Y1,Y2). && \text{(F.10)} \\
p(X,Y,X2,Y2) &\leftarrow \; p(X,Y,X1,Y1), cs(X1,A,X2), cs(Y1,A,Y2). && \text{(F.11)}
\end{aligned}
$$

$$p(X,Y,Y1,X1) \leftarrow p(X,Y,X1,Y1). \quad \text{(F.12)}$$
$$p(X,Y,X1,Y2) \leftarrow p(X,Y,X1,Y1), p(X,Y,Y1,Y2). \quad \text{(F.13)}$$
$$dist(X,Y) \leftarrow p(X,Y,X1,Y1), view(X1,V), not\ view(Y1,V). \quad \text{(F.14)}$$
$$dist(X,Y) \leftarrow p(X,Y,X1,Y1), not\ view(X1,V), view(Y1,V). \quad \text{(F.15)}$$
$$ceq(X,Y); \neg ceq(X,Y) \leftarrow . \quad \text{(F.16)}$$
$$\leftarrow not\ ceq(X,X). \quad \text{(F.17)}$$
$$\leftarrow ceq(X,Y), not\ ceq(Y,X). \quad \text{(F.18)}$$
$$\leftarrow ceq(X,Y), ceq(Y,Z), not\ ceq(X,Z). \quad \text{(F.19)}$$

$$\leftarrow ceq(X,Y), view(X,V), not\ view(Y,V). \quad \text{(F.20)}$$
$$\leftarrow ceq(X,Y), not\ view(X,V), view(Y,V). \quad \text{(F.21)}$$
$$\leftarrow not\ ceq(X1,Y1), ceq(X,Y), cs(X,A,X1), cs(Y,A,Y1). \quad \text{(F.22)}$$

$$\leftarrow not\ ceq(X,Y), not\ dist(X,Y). \quad \text{(F.23)}$$

where the variables X and Y variable range over distinctive states and the variable V ranges over views. Rule F.16 states that an answer set of the program should be *complete* with respect to *ceq*. Rules F.17-F.19 require *ceq* to be an equivalence class. Rules F.20 and F.21 are the counterpart of axiom F.6. Rule F.22 is the counterpart of axiom F.7. In order to define the maximality condition of *ceq*, the auxiliar predicate $p(X,Y,X1,Y1)$ is introduced. This predicate reads as *"If X and Y were the same, then $X1$ and $Y1$ would be the same"*. The predicate $dist(X,Y)$ defines when distinctive states X and Y are distinguishable. Constraint F.23 establishes the maximality condition on *ceq*: $ceq(X,Y)$ should be the case unless X and Y are distinguishable.

Notation. Given a set of experiences E, $\Pi(E)$ denotes the grounded program consisting of the rules $\{cs(ds,a,ds') \leftarrow . : E \models \langle ds,a,ds'\rangle\}$, $\{view(ds,v) \leftarrow . : E \models View(ds,v)\}$, and replacing in Π the occurrences of the variables X, $X1$, $X2$, Y, $Y1$, $Y2$ and V by distinctive states and view symbols in E, respectively.

Given a model M for the axioms F.1-F.7, $\Pi(E)(ceq^M)$ denotes the program consisting of $\Pi(E)$ and the rules $\{ceq(a,b) \leftarrow . : M \models ceq(a,b)\}$, $\{\neg ceq(a,b) \leftarrow . : M \not\models ceq(a,b)\}$.

Similarly, $\Pi(E)_1(ceq^M)$ denotes the program resulting by removing from $\Pi(E)(ceq^M)$ the rules associated with grounding rule F.23. By $AS(\Pi(E)_1(ceq^M))$ we denote the answer set of $\Pi(E)_1(ceq^M)$ (see lemma 2, page 239).

We say that ceq^M is *maximal* in M whenever M is a model for $CT(E)$.[1]
{*end of notation*}

Using the notation above we can state our theorem as follows:

Theorem 13 *ceq^M is* maximal *in M if and only if $AS(\Pi(E)_1(ceq^M))$ is an answer set for $\Pi(E)(ceq^M)$.*

Notice that theorem 13 establishes a one to one correspondence between the answer sets of $\Pi(E)$ and the models of $CT(E)$. Given a model M for $CT(E)$, ceq^M is *maximal* in M, and so $AS(\Pi(E)_1(ceq^M))$ is an answer set for $\Pi(E)(ceq^M)$, thus, an answer set for $\Pi(E)$.[2] Conversely, given any answer set X for $\Pi(E)$, the model M defined such that $M \models E$ and $ceq^M = \{(ds, ds') : ceq(ds, ds') \in X\}$, is such that $AS(\Pi(E)_1(ceq^M)) = X$.[3] Consequently, ceq^M is *maximal* in M, that is, M is a model for $CT(E)$.

Proof of theorem 13.

(a) Suppose ceq^M is maximal in M and $AS(\Pi(E)_1(ceq^E))$ is **not** an answer set for $\Pi(E)(ceq^M)$. Since $AS(\Pi(E)_1(ceq^M))$ is an answer set for $\Pi(E)_1(ceq^M)$, then $AS(\Pi(E)_1(ceq^M))$ does not satisfy constraint F.23.[4] Consequently, there exist distinctive states X and Y such that:

1. $ceq(X, Y) \notin AS(\Pi(E)_1(ceq^M))$, and
2. $dist(X, Y) \notin AS(\Pi(E)_1(ceq^M))$.

Define $ceq1^M$ in M such that $M \models ceq1(a, b)$ whenever $M \models ceq(a, b)$ or $p(X, Y, a, b) \in AS(\Pi(E)_1(ceq^M))$. Symbolically,

$$ceq1^M(a, b) \equiv ceq^M(a, b) \vee p(X, Y, a, b)\ .$$

By definition, $ceq^M \subset ceq1^M$.[5] We are to prove that $ceq1^M$ satisfies axioms F.3-F.7, which will contradict the fact that ceq^M is maximal in M:

[1]Recall that ceq^M denotes the interpretation of ceq in the structure M.

[2]Let X be an answer set for $\Pi(E)(ceq^M)$ and consider $Y \subseteq X$ closed under $\Pi(E)$. Then, in virtue of F.16, Y is closed under ceq^M and so $X = Y$.

[3]Given any program Π and X an answer set for Π, X is the unique answer set for $\Pi \cup X$.

[4]The answer sets of $\Pi(E)(ceq^M)$ are those answer sets of $\Pi(E)_1(ceq^M)$ that satisfy constraint F.23.

[5]Since $p(X, Y, X, Y) \in AS(\Pi(E)_1(ceq^M))$

- By lemma 3, $ceq1^M$ is an equivalence relation.
- Suppose that $ceq1^M(ds, ds')$ is the case. If $ceq^M(ds, ds')$ is the case, then $M \models View(ds, v) \equiv View(ds', v)$. If $p(X, Y, ds, ds')$ is the case, then $View(ds, v) \equiv View(ds', v)$ is the case, since otherwise $dist(X, Y)$ will belong to $AS(\Pi(E)_1(ceq))$.[6]
- Suppose that $M \models ceq1(ds_1, ds_2) \wedge \langle ds_1, a, ds'_1 \rangle \wedge \langle ds_2, a, ds'_2 \rangle$. If $M \models ceq(ds_1, ds_2)$ is the case, then $M \models ceq(ds'_1, ds'_2)$ is the case. If $p(X, Y, ds_1, ds_2)$ is the case, by rule F.11, $p(X, Y, ds'_1, ds'_2) \in AS(\Pi(E)_1(ceq^M))$ and so $M \models ceq1(ds'_1, ds'_2)$.

  □

(b) Suppose that $AS(\Pi(E)_1(ceq))$ is an answer set for $\Pi(E)(ceq^M)$ and ceq^M is *not* maximal in M. Then, there exists $ceq1^M$, $ceq^M \subset ceq1^M$, $ceq1^M$ maximal in M. Moreover, by the if part of this theorem, (a) above, $AS(\Pi(E)_1(ceq1^M))$ is an answer set for $\Pi(E)(ceq1^M)$. Let X and Y be such that:

1. $M \models ceq1(X, Y)$,
2. $M \not\models ceq(X, Y)$, and so (Lemma 2) $ceq(X, Y) \notin AS(\Pi(E)_1(ceq^M))$.

Since $AS(\Pi(E)_1(ceq^M))$ satisfies constraint F.23, then $dist(X, Y) \in AS(\Pi(E)_1(ceq^M))$, and consequently (rules F.14-F.15) there exist $X1$,$Y1$ and V, such that

1. $p(X, Y, X1, Y1) \in AS(\Pi(E)_1(ceq^M))$,
2. $M \models View(X1, V) \not\equiv View(Y1, V)$.

Since $ceq^M \subset ceq1^M$, then $p(X, Y, X1, Y1) \in AS(\Pi(E)_1(ceq1^M))$ (Lemma 4). Since $ceq(X, Y) \in AS(\Pi(E)_1(ceq1^M))$, then $ceq(X1, Y1) \in AS(\Pi(E)_1(ceq1^M))$ (Lemma 5) which is a contradiction since $AS(\Pi(E)_1(ceq1^M))$ satisfies constraints F.20-F.21.

□

Lemma 2 *Let M be a model for axioms F.1-F.7. Let ceq^M be the interpretation of ceq in M.*[7] *Then $\Pi(E)_1(ceq^M)$ has a unique answer set which we denote by $AS(\Pi(E)_1(ceq^M))$. Moreover, for any two distinctive states ds and ds', $M \models ceq(ds, ds')$ if and only if $ceq(ds, ds') \in AS(\Pi(E)_1(ceq^M))$.*

[6]This is the case according to rules F.14 and F.15, and the fact that $AS(\Pi(E)_1(ceq))$ is the answer set for $\Pi(E)_1(ceq)$.

[7]*The interpretation of ceq in M is not necessarily maximal.*

Proof. Let $\Pi(E)_2(ceq^M)$ denote the program resulting of removing from $\Pi(E)_1(ceq^M)$ those constraints resulting from grounding rules F.17-F.22. The answer sets of $\Pi(E)_1(ceq^M)$ are those answer sets of $\Pi(E)_2(ceq^M)$ satisfying constraints F.17-F.22. We are to prove that $\Pi(E)_2(ceq^M)$ has a unique answer set satisfying constraints F.8-F.15.

Let $Facts$ denote the union of the sets $\{cs(ds, a, ds') \leftarrow . : E \models \langle ds, a, ds'\rangle\}$, $\{view(ds, v) \leftarrow . : E \models View(ds, v)\}$, $\{ceq(a, b) \leftarrow . : M \models ceq(a, b)\}$, and $\{\neg ceq(a, b) \leftarrow . : M \not\models ceq(a, b)\}$. Any answer set of $\Pi(E)_2(ceq^M)$ contains $Facts$. Let X and Y denote two possible answer sets for $\Pi(E)_2(ceq^M)$. Then the reduct of X and Y are the same, $\Pi(E)_2(ceq^M)^X = \Pi(E)_2(ceq^M)^Y$, since both X and Y agree on the literals of the form $view(ds, v)$.[8] Consequently, $\Pi(E)_2(ceq^M)$ has at most one answer set. In fact, $Cn(\Pi(E)_2(ceq^M)^{Facts})$ is such answer set.[9] In particular, $ceq(ds, ds') \in Cn(\Pi(E)_2(ceq^M)^{Facts})$ iff $ceq(ds, ds') \in Facts$ iff $M \models ceq(ds, ds')$.

Finally, since ceq^M satisfies axioms F.3-F.7 then $Cn(\Pi(E)_2(ceq^M)^{Facts})$ satisfies constraints F.17-F.22, thus, it is an answer set for $\Pi(E)_1(ceq^M)$. □

Lemma 3 *Let M be a model for axioms F.1-F.7. Let ceq^M be the interpretation of ceq in M, and let $AS(\Pi(E)_1(ceq^M))$ be the answer set for $\Pi(E)_1(ceq^M)$. Let X and Y be two arbitrary distinctive state symbols. Let $ceq1^M$ in M be such that $M \models ceq1(a, b)$ whenever $M \models ceq(a, b)$ or $p(X, Y, a, b) \in AS(\Pi(E)_1(ceq^M))$. Symbolically,*

$$ceq1^M(a, b) \equiv ceq^M(a, b) \vee p(X, Y, a, b) \ .$$

Then, $ceq1^M$ is an equivalence relation.

Proof.

- $ceq1$ is reflexive. Indeed,

$$ceq1^M(ds, ds) \equiv ceq^M(ds, ds) \vee p(X, Y, ds, ds) \overset{F.3}{\equiv} ceq^M(ds, ds) \ .$$

- $ceq1$ is symmetric. Indeed,

$$\begin{array}{rcl} ceq1^M(ds, ds') & \equiv & ceq^M(ds, ds') \vee p(X, Y, ds, ds') \\ & \overset{F.4}{\equiv} & ceq^M(ds', ds) \vee p(X, Y, ds, ds') \\ & \overset{F.12}{\equiv} & ceq^M(ds', ds) \vee p(X, Y, ds', ds) \\ & \equiv & ceq1^M(ds1, ds) \end{array}$$

[8]Notice that $view(ds, ds') \in X$ iff $view(ds, ds') \in Facts$.

[9]$\mathbf{Cn(\Pi)}$ denotes the set of consequences of a logic program without negation as failure.

- $ceq1$ is transitive. Indeed,

$$
\begin{array}{ll}
\multicolumn{2}{l}{ceq1^M(ds,ds') \wedge ceq1^M(ds',ds'')} \\
\equiv & \{ceq^M(ds,ds') \vee p(X,Y,ds,ds')\} \wedge \{ceq^M(ds',ds'') \vee p(X,Y,ds',ds'')\} \\
\equiv & \{ceq^M(ds,ds') \wedge ceq^M(ds',ds'')\} \vee \{ceq^M(ds,ds') \wedge p(X,Y,ds',ds'')\} \vee \\
 & \{p(X,Y,ds,ds') \wedge ceq^M(ds',ds'')\} \vee \{p(X,Y,ds,ds') \wedge p(X,Y,ds',ds'')\} \\
\overset{F.5,F.13}{\rightarrow} & ceq^M(ds,ds'') \vee \{ceq^M(ds,ds') \wedge p(X,Y,ds',ds'')\} \vee \\
 & \{p(X,Y,ds,ds') \wedge ceq^M(ds',ds'')\} \vee p(X,Y,ds,ds'') \\
\overset{F.9,F.10}{\rightarrow} & ceq^M(ds,ds'') \vee p(X,Y,ds,ds'') \vee p(X,Y,ds,ds'') \vee p(X,Y,ds,ds'') \\
\equiv & ceq^M(ds,ds'') \vee p(X,Y,ds,ds'') \\
\equiv & ceq1^M(ds,ds'')
\end{array}
$$

□

Lemma 4 *Let M be a model for axioms F.1-F.7, and let ceq^M, $ceq1^M$, $ceq^M \subset ceq1^M$, be two relations such that both satisfy axioms F.3-F.7. Then, $AS(\Pi(E)_1(ceq^M)) \subset AS(\Pi(E)_1(ceq1^M))$.*

Proof. Let $\Pi(E)_2(ceq^M)$ denote the program resulting of removing from $\Pi(E)_1(ceq^M)$ those constraints resulting from grounding rules F.17-F.22. Let $Facts$ denote the union of the sets $\{cs(ds,a,ds') \leftarrow . : E \models \langle ds,a,ds'\rangle\}$, $\{view(ds,v) \leftarrow . : E \models view(ds,v)\}$, $\{ceq(a,b) \leftarrow . : M \models ceq(a,b)\}$, and $\{\neg ceq(a,b) \leftarrow . : M \not\models ceq(a,b)\}$. Define $Facts1$ in the same way as $Facts$ but using $ceq1$ instead of ceq. Since, $ceq^M \subset ceq1^M$, it is the case that $Facts \subset Fact1$ and consequently $\Pi(E)_2(ceq^M) \subset \Pi(E)_2(ceq1^M)$.

Since, $Facts$ and $Facts1$ agree on literals of the form $view(ds,v)$, it follows that $\Pi(E)_2(ceq^M)^{Facts} \subset \Pi(E)_2(ceq1^M)^{Facts1}$. It follows then that $Cn(\Pi(E)_2(ceq^M)^{Facts}) \subset Cn(\Pi(E)_2(ceq^M)^{Facts1}) \subset Cn(\Pi(E)_2(ceq1^M)^{Facts1})$. In lemma 2 we prove that $AS(\Pi(E)_1(ceq^M)) = Cn(\Pi(E)_2(ceq^M)^{Facts})$ and $AS(\Pi(E)_1(ceq1^M)) = Cn(\Pi(E)_2(ceq^M)^{Facts1})$. Therefore, $AS(\Pi(E)_1(ceq^M)) \subset AS(\Pi(E)_1(ceq1^M))$.

□

Lemma 5 *Let M be a model for axioms F.1-F.7. If $ceq^M(ds,ds')$ and $p(ds,ds',ds1,ds1') \in AS(\Pi(E)_1(ceq^M))$, then $ceq^M(ds1,ds1')$.*

Proof. Consider the program $\hat{\Pi}(E)_1(ceq^M)$ resulting from $\Pi(E)_1(ceq^M)$, by removing those instances of rules F.12 and F.13 where $X = ds$ and $Y = ds'$. Let AS denote the answer set for $\hat{\Pi}(E)_1(ceq^M)$. We are to prove that $AS(\Pi(E)_1(ceq^M)) = AS$, which in virtue of property F.24, proves our theorem. Notice that $AS(\Pi(E)_1(ceq^M))$ is closed under $\hat{\Pi}(E)_1(ceq^M))$, and consequently $AS \subseteq AS(\Pi(E)_1(ceq^M))$. We need to prove then that AS is closed under $\Pi(E)_1(ceq^M))$.

Since AS is the answer set for $\hat{\Pi}(E)_1(ceq^M)$, $p(ds, ds', X, Y) \in AS$ if and only if there exist distinctive states $X_0, \hat{X}_0, Y_0, \hat{X}_0, \ldots, X_n, \hat{X}_n, Y_n, \hat{X}_n$ and actions $A_0, \ldots, A_{n-1}$ such that

1. $(X_0, Y_0) = (ds, ds')$, $(\hat{X}_n, \hat{Y}_n) = (X, Y)$,
2. $ceq(X_i, \hat{X}_i)$, $ceq(Y_i, \hat{Y}_i)$, $0 \leq i \leq n$.
3. $cs(\hat{X}_i, A_i, X_{i+1})$, $cd(\hat{Y}_i, A_i, Y_{i+1})$, $0 \leq i < n$.

By induction on n we can prove then that

$$if\ p(ds, ds', X, Y) \in AS\ then\ ceq(X, Y) \in AS. \tag{F.24}$$

For $n = 0$ we have that $(X_0, Y_0) = (ds, ds')$, $(\hat{X}_0, \hat{Y}_0) = (X, Y)$, $ceq(ds, X)$, and $ceq(ds', Y)$. Since ceq is an equivalence relation and $ceq(ds, ds')$, it follows that $ceq(X, Y)$. Suppose now that $(\hat{X}_n, \hat{Y}_n) = (X, Y)$ for some $n > 0$. By induction hypothesis, $ceq(\hat{X}_{n-1}, \hat{Y}_{n-1})$. Since ceq satisfies constraint F.22, it follows that $ceq(X_n, Y_n) \in AS$. Since ceq is an equivalence relation, it follows then that $ceq(\hat{X}_n, \hat{Y}_n)$.

Using F.24 we prove that AS is closed under $\Pi(E)_1(ceq^M))$. Indeed,

$$\begin{aligned} p(ds, ds', Y, X) &\overset{F.10}{\leftarrow} p(ds, ds', Y, Y), ceq(Y, X) \\ &\overset{F.9,F.4}{\leftarrow} p(ds, ds', X, Y), ceq(X, Y) \\ &\overset{F.24}{\leftarrow} p(ds, ds', X, Y) \in AS \quad . \end{aligned}$$

$$\begin{aligned} p(ds, ds', X, Z) &\overset{F.9}{\leftarrow} p(ds, ds', Y, Z) \wedge ceq(Y, X) \\ &\overset{F.5}{\leftarrow} p(ds, ds', X, Y) \wedge ceq(X, Y) \wedge \\ &\qquad\qquad p(ds, ds', Y, Z) \wedge ceq(Y, Z) \\ &\overset{F.24}{\leftarrow} p(ds, ds', X, Y) \in AS \wedge p(ds, ds', Y, Z) \in AS \quad . \end{aligned}$$

We have proved that AS is closed under $\Pi(E)_1(ceq^M))$. It follows that $AS = AS(\Pi(E)_1(ceq^M))$. $\Box$

Appendix G

MERCATOR's Ontology and Semantics

MERCATOR [Davis, 1993] uses $clumps$, $regions$, $polygons$, $joints$, $edges$, $vertices$ and $PCOs$.[1] The basic elements of the representation are straight line segments. The boundary of an object is represented by a set of *edges* connecting *vertices*. Boundary edges are labeled with directions specifying the direction counter-clockwise around the object. Such a directed edge is called a *bound*. The interior of an object is represented by *polygons*. A complete shape description, consisting of a set of bounds and a set of polygons, is called a *region*.

Local dimensions are recorded in terms of the lengths and orientations of edges connecting the vertices. Lengths and orientations are not specified precisely. Rather, the range in which they lie is specified. The measure of the inaccuracy of a region is its *grain-size*, which is an upper bound on the distance from any point in the region to a point in the object. The smaller the grain-size, the better the approximation. Also, every bound in a region has a grain-size which, roughly speaking, is an upper bound on the distance from the bound to the corresponding part of the boundary.

The overall representation of an object is called a $clump$; it contains all the regions of the object, plus descriptions of the properties of the object, and the relations between the regions. A map is hierarchically arranged by containment. Clumps point to their immediate containers and contents.

[1]PCO stands for Partial Circular Ordering.

Different regions for a clump can be related to one another in three ways. Firstly, different regions may share one or more edges. Secondly, vertices of different edges may be connected by edges or joints. Thirdly, the order of the external vertices around the boundary is recorded in *partial circular ordering (PCO)*. Given two vertices, a PCO expresses whether they are in clockwise order, counter clockwise order, or unordered.

An *object* O is a closed, connected subset of R^2 (the real plane) with a boundary consisting of a finite number of disjoint simple closed curves. A *property* is a function from objects to arbitrary sets. A MERCATOR map describes a set of objects with properties.

Three functions relate the MERCATOR map to the real world: REAL, COOR, and COVER. REAL maps clumps onto objects. REAL preserves containment (i.e. if CL1 contains CL2, then REAL(CL1) must contain REAL(CL2)) and it takes clumps with stated properties onto objects with these properties. COOR takes vertices of the map into points in the plane (even if a point g represents an object with some extent, COOR (g) is a single point in the plane). COOR is extended in the natural way to take edges into line segments, joints into pairs of line segments, polygons of the map into planar polygons, and regions into unions of polygons. COOR has to satisfy the following conditions:

1. For each edge e, $COOR(e)$ has to have length and orientation within the fuzz ranges which the map specifies for e. Likewise for joints.
2. For each polygon P in the map, $COOR(P)$ must define a polygon in the plane.
3. For each region R, every point in $COOR(R)$ must be no further than the grain-size of the interior of R from the object represented by R.

COVER is a family of functions. COOR, as stated, maps directed edges onto line segments in the plane. However, we must relate boundary edges to the part of the object boundary that they represent. Therefore, for each directed edge b in the boundary of a region, we define a continuous function $COVER_b$ from $COOR(b)$ into the object boundary. $COVER_b$ must satisfy the following conditions:

1. For all $x \in COOR(b)$, the distance from x to $COVER_b(x)$ must be less than the grain-size of b.
2. If b and c are directed edges from the shell of region R which meet at a vertex v, and v is a real bound of R, then $COVER_b(COOR(v)) = COOR_c(COOR(v))$.
3. The real vertices of a clump map into the boundary of the corresponding object so as to satisfy the PCOs of the clump.

The MERCATOR map is valid if it is possible to define REAL, COOR, and COVER so as to satisfy all these conditions.

Bibliography

[Angluin, 1978] Dana Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 39:337–350, 1978.

[Angluin, 1987] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.

[Anousaki and Kyriakopoulos, 1999] G.C. Anousaki and T. Kyriakopoulos. Simultaneous localization and map building for mobile robot navigation. *IEEE Robotics & Automation Magazine*, pages 42–53, 1999.

[Aurenhammer, 1991] F. Aurenhammer. Voronoi diagrams– a survey of fundamental geometric structure. *ACM Computing Surveys*, 23:345–405, 1991.

[Bacchus *et al.*, 1999] F. Bacchus, J. Halpern, and H. Levesque. Reasoning about noisy sensors and effectors in the situational calculus. *Artificial Intelligence*, 111:171–208, 1999.

[Baral and Gelfond, 1997] Chitta Baral and Michael Gelfond. Reasoning about effects of concurrent actions. *Journal of Logic Programming*, 31(1-3):85–117, 1997.

[Baral and Gelfond, 1999] Chitta Baral and Michael Gelfond. Reasoning agents in dynamic domains. In Jack Minker, editor, *Workshop on Logic-Based Artificial Intelligence, Washington, DC, June 14–16, 1999*, College Park, Maryland, 1999. Computer Science Department, University of Maryland.

[Basye *et al.*, 1995] K. Basye, T. Dean, and L.P. Kaelbling. Learning dynamics: System identification for perceptually challenged agents. *Artificial Intelligence*, 72(1):139–171, 1995.

[Beardon, 1979] A. F. Beardon. *Complex Analysis*. John Weley & Sons, New York, 1979.

[Bickhard M., 1995] Terveen L. Bickhard M. *Foundational issues in Artificial Intelligence and Cognitive Science: Impasse and Solution*. North Holland, Elsevier Science B.V., first edition, 1995.

[Borenstein *et al.*, 1996] J. Borenstein, H.R. Everett, and L. Feng. *Navigating mobile robots: systems and techniques*. A K Peters, Wellesley, Massachusetts, 1996.

[Borestein and Koren, 1991] J. Borestein and Y. Koren. The vector field histogram - fast obstacle-avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 7(3):278–288, 1991.

[Borg and Groenen, 1997] Ingwer Borg and Patrick Groenen. *Modern multidimensional scaling: theory and applications*. Springer, New York, 1997.

[Choset and Nagatani, 2001] H. Choset and K. Nagatani. Topological simultaneous localization and mapping (SLAM): toward exact localization without explicit localization. *IEEE Trans. on Robotics and Automation*, 17(2):125–137, April 2001.

[Choset *et al.*, 1997] H. Choset, I. Konuksven, and A. Rizzi. Sensor based planning: A control law for generating the generalized voronoi diagram. *IEEE International Conference in Autonomous Robots*, 1997.

[Cormen *et al.*, 1993] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT press, Cambridge, Massachusetts, 1993.

[Crawford and Kuipers, 1991] J. Crawford and B. Kuipers. Algernon: a tractable system for knowledge representation. *SIGART Bulleting*, 2(3):35–44, 1991.

[Davis, 1993] E. Davis. The Mercator representation of spatial knowledge. In *AAAI-93*, 1993.

[De Giacomo *et al.*, 1997] G. De Giacomo, Y. Lesperance, and H. Levesque. Reasoning about concurrent execution, prioritized interrupts, and exogenous actions in the situation calculus. In *IJCAI 97, Nagoya*, pages 1221–1226, 1997.

[Duboi and Prade, 1988] D. Duboi and H. Prade. Representation and combination of uncertainty with belief functions and possibility measures. *Computational Intelligence*, 4:244–264, 1988.

[Duckett and Nehmzow, 2000] Tom Duckett and Ulrich Nehmzow. Performance comparison of landmark recognition systems for navigating mobile robots. In *AAA-I 2000*, pages 826–831, July-August 2000.

[Dudek *et al.*, 1991] Gregory Dudek, Jenkin Michael, Evangelos Milios, and David Wilkes. Robotic exploration as graph construction. *IEEE Trans. on Robotics and Automation*, 7(6):859–865, 1991.

[Durrant-Whyte, 1987] H. F. Durrant-Whyte. Consistent integration and propagation of disparate sensor observations. *The International Journal of Robotics Research*, 6(3):3–24, 1987.

[Durrant-Whyte, 1988a] H. F. Durrant-Whyte. *Integration, coordination and control of multisensor robot systems*. Kluwer Academic Publishers, Boston, 1988.

[Durrant-Whyte, 1988b] H. F. Durrant-Whyte. Uncertain geometry in robotics. *IEEE Journal of Robotics and Automation*, 5(6):23–31, 1988.

[Elfes, 1987] Alberto Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, RA3(3):249–265, 1987.

[Engelson and McDermott, 1992a] Sean P. Engelson and Drew V. McDermott. Maps considered as adaptive planning resources. In *AAAI Fall Symposium on Applications of AI to Real-World Autonomous Mobile Robots, Working Notes*, Cambridge, MA, October 1992.

[Engelson and McDermott, 1992b] S.P. Engelson and D.V. McDermott. Error correction in mobile robot map learning. In *the IEEE International Conference on Robotics and Automation*, pages 2555–2560, 1992.

[Fernandez and Gonzalez, 1997] Juan Fernandez and Javier Gonzalez. A general world representation for mobile robot operations. In *Seventh conference of the Spanish association for artificial intelligence (CAEPIA-97)*, 1997.

[Fernandez and Gonzalez, 1998] Juan Fernandez and Javier Gonzalez. Hierarchical graph search for mobile robot planning. In *IEEE international conference on Robotics and Automation*, 1998.

[Fernandez, 2000] Juan Fernandez. *Modeling and Generation of Multiple Abstractions for Representing Large-Scale Space. An Application to Mobile Robots.* PhD thesis, University of Malaga, Spain, 2000.

[Fikes and Nilsson, 1971] R. Fikes and N. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Firby, 1994] R. James Firby. Task networks for controlling continuous processes. In Kristian Hammond, editor, *Second International Conference on AI Planning Systems*, pages 49–54. AAAI Press, 1994.

[Franz *et al.*, 1998] M. Franz, B. Schölkopf, H. Mallot, and Bülthoff. Learning view graphs for robot navigation. *Autonomous robots*, 5:111–125, 1998.

[Freksa, 1992] C. Freksa. Using orientation information for qualitative spatial reasoning. *Theories and Models of Spatio-Temporal Reasoning in Geographic Space*, 639:162–178, 1992.

[Gat, 1996] Erann Gat. ESL: A language for supporting robust plan execution in embedded autonomous agents. In *Plan Execution: Problems and Issues: Papers from the 1996 AAAI Fall Symposium*, pages 59–64. AAAI Press, Menlo Park, California, 1996.

[Gelb, 1974] A. Gelb. *Applied Optimal Estimation*. MIT Press, Cambridge, MA, 1974.

[Gelfond and Lifschitz, 1991] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[Gelfond and Lifschitz, 1998] M. Gelfond and V. Lifschitz. Action Languages. *Linköping electronic articles in computer and information sciences*, 3(16), 1998.

[Gershenfeld, 1999] Neil Gershenfeld. *The Nature of Mathematical Modeling*. Cambridge University Press, 1999.

[Giunchiglia and Lifschitz, 1998] E. Giunchiglia and V. Lifschitz. An action laguage based on causal explanation: preliminary report. In *AAAI-98*, pages 623–630, 1998.

[Gold, 1978] E. Mark Gold. Complexity of automaton identification from given sets. *Information and Control*, 37:302–320, 1978.

[Gonzalez and Woods, 1992] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Addison-Wesley Publishing Company, 1992.

[Gopal and Smith, 1990] S. Gopal and T.R. Smith. Human way-finding in an urban environment: a performance analysis of a computational process model. *Environment and Planning*, 22:169–191, 1990.

[Gopal *et al.*, 1989] S. Gopal, R. L. Klatzky, and T. R. Smith. Navigator: a psychologically based model of environmental learning through navigation. *Journal of Environmental Psychology*, 9:309–331, 1989.

[Green, 1969] C. Green. Applications of theorem proving to problem solving. In *IJCAI-69*, pages 219–240, 1969.

[Hernandez, 1994] D. Hernandez. *Qualitative Representation of Spatial Knowledge*. Number 804 in Lecture Notes in Computer Science. Springer-Verlag, 1994.

[J., 1991] Levenick J. NAPS: A connectionist implementation of cognitive maps. *Connection Science*, 3:107–126, 1991.

[Kay *et al.*, 1999] Herbert Kay, Bernard Rinner, and Benjamin Kuipers. Semi-quantitavive system identification. Technical Report AI99-279, Computer Science Department, University of Texas at Austin, March 1999.

[Koenig and Simmons, 1996] S. Koenig and Reid Simmons. Passive distance learning for robot navigation. In *Proceedings ofthe Thirteenth International Conference on Machine Learning (ICML)*, pages 266 – 274, 1996.

[Koening and Simmons, 1998] S. Koening and R.G. Simmons. XAVIER: A robot navigation architecture based on partially observable markov decision process models. In *Artificial Intelligence and Mobile Robots*, pages 91–122. MIT press, 1998.

[Konolige *et al.*, 1995] K. Konolige, A. Saffiotti, and E. Ruspini. A multivalued logic approach to integrating planning and control. *Artificial Intelligence*, 76:481–526, 1995.

[Kortenkamp *et al.*, 1995] D. Kortenkamp, E. Chown, and S. Kaplan. Prototypes, locations, and associative networks (PLAN): towards a unified theory of cognitive mapping. *Cognitive Science*, 19:1–51, 1995.

[Kortenkamp *et al.*, 1998] D. Kortenkamp, R.P. Bonasso, and R. Murphy. *Artificial Intelligence and Mobile Robots*. AAAI Press, 1998.

[Kuipers and Byun, 1988] B. Kuipers and Y. T. Byun. A robust qualitative method for spatial learning in unknown environments. In Morgan Kaufmann, editor, *AAAI-88*, 1988.

[Kuipers and Byun, 1991] B. Kuipers and Y. T. Byun. A robot exploration and mapping strategy based on semantic hierarchy of spatial representations. *Journal of Robotics and Autonomous Systems*, 8:47–63, 1991.

[Kuipers and Levitt, 1988] Benjamin. Kuipers and T. Levitt. Navigation and mapping in large-scale space. *AI Magazine*, 9(2):25–43, 1988.

[Kuipers *et al.*, 1993] B. Kuipers, R. Froom, W.Y. Lee, and D. Pierce. The semantic hierarchy in robot learning. In J. Connell and S. Mahadevan, editors, *Robot Learning*, pages 141–170. Kulwer Academic Publishers, 1993.

[Kuipers, 1978] Benjamin Kuipers. Modeling spatial knowledge. *Cognitive Science*, 2:129–153, 1978.

[Kuipers, 1996] Benjamin Kuipers. A hierarchy of qualitative representations for space. In *Working papers of the Tenth International Workshop on Qualitative Reasoning about Physical Systems (QR-96)*. AAAI Press, 1996.

[Kuipers, 2000] Benjamin Kuipers. The spatial semantic hierarchy. *Artificial Intelligence Journal*, 119:191–233, 2000.

[Kuo, 1987] B. C. Kuo. *Automatic Control Systems*. Prentice-Hall, Inc., fifth edition, 1987.

[Lee, 1996] W. Y. Lee. *Spatial Semantic Hierarchy for a Physical Mobile Robot*. PhD thesis, The University of Texas at Austin, 1996.

[Leiser and Zilbershatz, 1989] David Leiser and Avishai Zilbershatz. THE TRAVELLER: a computational model of spatial network learning. *Environment and Behavior*, 21(4):435–463, 1989.

[Lesperance *et al.*, 1994] Y. Lesperance, H. Levesque, F. Lin, D. Marcu, R. Reiter, and R. Scherl. A logical approach to high level robot programming: a progress report. In Kuipers B., editor, *Control of the physical world by intelligent systems: papers from the 1994 AAAI fall symposium*, pages 79–85, 1994.

[Levitt and Lawton, 1990] T. S. Levitt and D. T. Lawton. Qualitative navigation for mobile robots. *Artificial intelligence*, 44(3):305–360, 1990.

[Lifschitz, 1994] Vladimir Lifschitz. Circumscription. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 3, pages 297–352. Oxford University Press, 1994.

[Lifschitz, 1995] Vladimir Lifschitz. Nested abnormality theories. *Artificial Intelligence*, (74):351–365, 1995.

[Lifschitz, 1999] V. Lifschitz. Answer set planning. *International Conference on Logic Programming*, 9:23–37, 1999.

[Lobo *et al.*, 1997] J. Lobo, S. Taylor, and G. Mendez. Adding knowledge to the action description language A. In *AAAI-97*, pages 454–459, 1997.

[Mallory *et al.*, 1996] R. Mallory, B. Porter, and B. Kuipers. Comprehending complex behavior graphs through abstraction. In *Working papers of the tenth international workshop on qualitative reasoning about physical systems (QR-96)*. AAAI press, 1996.

[Mallot and Gillner, 2000] H. A. Mallot and S. Gillner. Route navigating without place recognition: What is recognized in recognition-triggered responses? *Perception*, 29:43–55, 2000.

[Markman, 1999] A. B. Markman. *Knowledge Representation*. Lawrence Erlbaum Associates, 1999.

[McCain, 1999] Norman McCain. The causal calculator. Technical report, Computer Science Department, University of Texas at Austin, 1999.

[McCarthy and Buvač, 1998] John McCarthy and Sacha Buvač. Formalizing context (expanded notes). In A. Aliseda, R.J. van Glabbeek, and C. Westerstähl, editors, *Computing Natural Language*, volume 8L of CSLI Lecture Notes, pages 13–50. 1998.

[McCarthy and Hayes, 1969] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence*, volume 4. Edinburgh University Press, 1969.

[McCarthy, 1980] John McCarthy. Circumscription - A Form of Non-Monotonic Reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[McCarthy, 1986] John McCarthy. Applications of circumscription to formalizing commonsense knowledge. *Artificial Intelligence*, 3(26):88–116, 1986.

[McDermott and Davis, 1984] D. V. McDermott and E. Davis. Planning routes through uncertain territory. *Artificial Intelligence*, 22:107–156, 1984.

[McDermott, 1980] D. McDermott. A theory of metric spatial inference. In *AAAI-80*, pages 246–248, 1980.

[Moore, 1979] R.C. Moore. *Reasoning about knowledge and action*. PhD thesis, MIT, 1979.

[Moore, 1985] R.C. Moore. A formal theory of knowledge and action. In J.R. Hobbs and R.C. Moore, editors, *Formal Theories of the Common Sense World*, pages 319–358. Ablex Publishing, 1985.

[Moutarlier and Chatila, 1989] P. Moutarlier and R. Chatila. Stochastic multisensory data fusion for mobile robot location and environment modelling. In *5th International Symposium on Robotics Research*, pages 85–89, 1989.

[Niemelä and Simons, 1997] I. Niemelä and P. Simons. Smodels - an implementation of the stable model and well-founded semantics for normal logic programs. In *4th International Conference on Logic Programming and Nonmonotonic Reasoning*, number 1265 in Lecture Notes in Computer Science, pages 420–429. Springer-Verlag, 1997.

[Nourbakhsh and Genesereth, 1996] I. Nourbakhsh and M. Genesereth. Assumptive Planning and Execution: a Simple, Working Robot Architecture. *Autonomous Robots*, 3:49–67, 1996.

[Nourbakhsh, 1998] I. Nourbakhsh. DERVISH: An office-navigating robot. In *Artificial Intelligence and Mobile Robots*, pages 73–91. MIT press, 1998.

[O'Neill, 1991] Michael O'Neill. A biologically based model of spatial cognition and wayfinding. *Journal of Environmental Psychology*, 11:299–320, 1991.

[Peressini *et al.*, 1988] Antony L. Peressini, Francis E. Sullivan, and K. Jerry Uhl. *The mathematics of nonlinear programming*. Springer-Verlag, New York, 1988.

[Puterman, 1994] Martin L. Puterman. *Markov Decision Processes*. John Wiley and Sons, 1994.

[Reiter, 1996] R. Reiter. Natural actions, concurrency and continuous time in the situational calculus. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR'96)*, 1996.

[Remolina and Kuipers, 1998a] Emilio Remolina and Benjamin Kuipers. Boundary region relations. In *Cognitive Robotics, Papers from the 1998 AAAI Fall symposium, FS-98-02*, pages 117–124. AAAI press, Menlo Park, 1998.

[Remolina and Kuipers, 1998b] Emilio Remolina and Benjamin Kuipers. Towards a formalization of the Spatial Semantic Hierarchy. In *Fourth Symposium on Logical Formalizations of Commonsense Reasoning, London*, January 1998.

[Remolina *et al.*, 1999] Emilio Remolina, Juan Fernandez, Benjamin Kuipers, and Javier Gonzalez. Formalizing regions in the spatial semantic hierarchy: an AH-graphs implementation approach. In *Fourth international conference on Spatial Information Theory (COSIT'99)*, 1999.

[Rinner and Kuipers, 1999] B. Rinner and B. Kuipers. Monitoring piecewise continuous behaviors by refining trackers and theirs models. In *Sixteenth International Joint Conference on Artificial Intelligence*, pages 1191–1198, August 1999.

[Rivest and Schapire, 1987] Ronald L. Rivest and Robert E. Schapire. A new approach to unsupervised learning in deterministic environments. In *Proceedings of the Fourth International Workshop on Machine Learning*, 1987.

[Sandewall, 1994] E. Sandewall. *Features and Fluents: the representation of knowledge about dynamical systems*. Claredon Press, Oxford, 1994.

[Sandewall, 1996] E. Sandewall. Towards the validation of high-level action descriptions from their low-level definitions. *Linköping electronic articles in computer and information sciences*, 1(4):http://www.ep.liu.se/ea/cis/1996/004, 1996.

[Sandewall, 1997] E. Sandewall. Logic-based modelling of goal-directed behavior. *Linköping electronic articles in computer and information sciences*, 2(19):http://www.ep.liu.se/ea/cis/1997/019, 1997.

[Schölkopf and Mallot, 1995] B. Schölkopf and H. Mallot. View-based cognitive mapping and path planning. *Adaptative Behavior*, 3:311–348, 1995.

[Scherl and Levesque, 1993] R.B. Scherl and H. Levesque. The frame problem and knowledge-producing actions. In *AAAI-93*, pages 689–695, 1993.

[Shafer, 1976] G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, Princeton, NJ, 1976.

[Shanahan, 1996] M. P. Shanahan. Noise and the common sense informatic situation for a mobile robot. In *AAAI-96*, pages 1098–1103, 1996.

[Shanahan, 1997a] M. P. Shanahan. Noise, non-determinism and spatial uncertainty. In *AAAI-97*, pages 153–158, 1997.

[Shanahan, 1997b] M. P. Shanahan. *Solving the Frame problem: a mathematical investigation of the Common sense law of Inertia*. MIT Press, 1997.

[Shanahan, 1998] M. P. Shanahan. Reinventing Shakey. In *Cognitive Robotics, papers from the 1998 AAAI Fall symposium, FS-98-02*, pages 125–135. AAAI press, Menlo Park, 1998.

[Shatkay and Kaelbling, 1997] H. Shatkay and L. Kaelbling. Learning topological maps with weak local odometry information. In *IJCAI-97*, 1997.

[Simmons and Koening, 1995] R. Simmons and S. Koening. Probabilistic Robot Navigation in Partially Observable Environments. In *IJCAI 95*, 1995.

[Smith and Cheeseman, 1986] R. Smith and P. Cheeseman. On the representation of and estimation of spatial uncertainty. *The International Journal of Robotics Research*, 5:56–68, 1986.

[Son and Baral, 2001] T.C. Son and C. Baral. Formalizing Sensing Actions: a transition function based approach. *Artificial Intelligence*, pages 19–91, 2001.

[Steck and Mallot, 2000] S. D. Steck and H. A. Mallot. The role of global and local landmarks in virtual environment navigation. *Presence*, 9(1):69–83, 2000.

[Stevens and Coupe, 1978] A. L. Stevens and P. Coupe. Distortions in judged spatial relations. *Cognitive Psychology*, 10:422–437, 1978.

[Stevens, 1946] S. S. Stevens. On the theory of scales of measurement. *Science*, 103:677–680, 1946.

[Taylor, 1997] John R. Taylor. *An introduction to error analysis*. University Science Books, 1997.

[Thrun *et al.*, 1998] S. Thrun, S. Gutmann, D. Fox, and B. Burgard, W.and Kuipers. Integrating topological and metric maps for mobile robot navigation: A statistical approach. In *AAAI-98*, pages 989–995, 1998.

[Thrun, 1998] Sebastian Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, 1998.

[Vandorpe *et al.*, 1996] J. Vandorpe, H. Van Brussel, and H. Xu. Exact dynamic map building for a mobile robot using geometrical primitives produced by a 2d range finder. In *1996 IEEE International Conference on Robotics and Automation*, pages 901–908, April 1996.

[Yeap, 1988] W. K. Yeap. Towards a computational theory of cognitive maps. *Artificial Intelligence*, 34:297–360, 1988.

Vita

Emilio Remolina was born in Málaga, Colombia on May 7, 1968, the son of Pablo Remolina and Victoria Angarita de Remolina. After completing his high school work at the Colegio Nacional Custodio Garcia Rovira, Málaga, in 1985, he entered the Universidad de Los Andes in Bogotá, Colombia. He received the degrees of B.A. in Computer Science,December 1989, B.A. in Mathematics, December 1990, and M.A. in Mathematics, June 1993, from the Universidad de Los Andes. In September of 1993, he entered the Graduate School of the University of Texas at Austin. He received the degree of M.S. in Computer Science in December 1995. His work has concentrated on the area of logic artificial intelligence with applications to autonomous agents. His publications include:

- *A logical Account of Causal and Topological Maps*, Remolina E. and Kuipers B., International Joint Conference in Artificial Intelligence (IJCAI-01), pages 5-11, Seattle, August, 2001.
- *Getting to the airport: the oldest planning problem in AI*, Lifschitz V., McCain N., Remolina E. and Tacchella A., in book Logic-Based Artificial Intelligence, pages 147-165, 2000.
- *Formalizing regions in the Spatial Semantic Hierarchy: an AH-graphs implementation approach*, Remolina E., Fernandez J., Kuipers B. and Gonzalez J., International conference on Spatial Information theory (COSIT'99), Germany, 1999.
- *Boundary Region Relations*, Remolina E. and Kuipers B., AAAI Fall Symposium on Cognitive Robotics, Orlando, October 1998.
- *Towards a formalization of the Spatial Semantic Hierarchy*, Remolina E. and Kuipers B., Fourth Symposium on Logical Formalizations of Commonsense Reasoning, London, January 1998.

- *Integrating Vision and Spatial Reasoning for Assistive Navigation*, Gribble W., Browning R., Hewett M., Remolina E. and Kuipers B., in Assistive Technology and Artificial Intelligence, LNCS, 1998.

Permanent Address: 1660 S. Amphlett Blvd., Suite 350, San Mateo, CA 94402

This dissertation was typeset with LaTeX 2_ε[2] by the author.

[2]LaTeX 2_ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this dissertation were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin.