Formal explorations of knowledge intensive tasks
Jansen, M.G.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.

Download date: 13 Dec 2018
We present a framework for describing problem solving for knowledge intensive tasks in terms of dynamic change of knowledge. The aim is to provide a knowledge level description of what an agent knows before, during and after problem solving. Knowledge is taken here as a semantic notion related to the information-theoretic analysis of Dretske [23]. The formal layer of our representation is formed by update semantics [95]. We use (and interpret) this formalism to represent knowledge about possible solutions by means of knowledge states. Problem solving is described in terms of knowledge states, updates and epistemic tests. We distinguish between reactive and proactive updates, the use of memory and the specification of problem solving steps. Finally, we provide an example of how three classification criteria can be represented and compared within this framework.

6.1 Introduction

Knowledge intensive problems can often be classified as belonging to a certain type or task. Alternatively, some problems first have to be decomposed into several subproblems which each can be classified. The different tasks found in the literature [71, 87, 82] have surfaced over the years in knowledge engineering research.

With each task a set of Problem Solving Methods (PSMs) can be associated. Like tasks, PSMs are pragmatic rather than formal constructs. They can be seen as generalized methods of problem solving for a range of problems, belonging to the same task. Building knowledge intensive systems can benefit considerably from using available PSMs after the problem type has been established.

Since PSMs are generalized methods they often have to be adapted to the peculiarities of the problem at hand. From a formal point of view this pragmatic nature of PSMs and tasks leaves considerable room for vagueness. Logical properties of different problem types are often ignored. This becomes an issue when one tries to specify PSMs in a more precise way. For example, attempts have been made, and are still under development [7] to systematically compile the precise descriptions of problems that are covered by a task description. The aim of those endeavours is to categorize problem solving methods into libraries and make them available for large-scale reuse.

Both the goal of a task and its PSMs are usually described using vocabulary defined in a task ontology. The methods of a task operate on domain knowledge which is defined in a separate domain ontology. In addition a method has a typical control structure which defines its data flow.
Figure 6.1

A partial mapping between problem solving methods and epistemic change is shown. Problem solving methods will be described in terms of epistemic change. Knowledge states will capture both the dynamic and static knowledge of an agent. The flow of knowledge will be described in terms of updates and tests on these knowledge states.

The description of tasks in our framework will assume the existence of both task and domain ontologies. The task ontology will consist of formal definitions of the concepts used in the descriptions of the task. The domain ontology should do likewise for the domain knowledge. To this we add the distinction that some knowledge remains invariant or static during problem solving, whereas some will be subject to change and is dynamic. The main focus will be on the dynamic characterization of the knowledge flow.

Problem solving will be treated as an attempt to acquire knowledge about possible solutions. Our view on knowledge is in line with the information theoretic notion as described in Dretske [23]. It is also similar to the one described in Fagin et al [24] for multi-agent systems.

The information theoretic account of knowledge links the amount of knowledge to the number of possible states of affairs. A knowledge-level agent has no knowledge if all possibilities are open. His knowledge increases when the number of possibilities decreases. We present a semantic framework of epistemic change, based on an information theoretic view on knowledge, and use it to describe problem solving behaviour.

The formal heart of our framework is formed by update semantics as described by Veltman [95]. It allows for standard as well as non-standard reasoning to be cast in terms of information states, updates and epistemic tests. However, we present a particular interpretation of update semantics. We will use information states as knowledge or solution states. They should capture the knowledge an agent has during each stage of the problem solving process. State transition takes place by processing new information in the form of updates. Epistemic tests enable one to verify properties of a state without state change.

Hence, we use update semantics to describe changing knowledge during problem solving. For this purpose we introduce the distinction between reactive and proactive updates. Reactive updates describe changes in knowledge of the agent induced by the environment. Proactive updates reflect decisions made by the agent itself and make use of a memory. Problem solving can then be described as a series of updates and tests on successive knowledge states. This flow of knowledge will be illustrated by diagrams.

The focus of our description will not be on the specification of efficient methods, or on the complexity of problems, but rather on a description of what knowledge an agent has before, dur-
ing and after solving a problem. Specifications will be given in terms of knowledge-level [67] constructs, such as knowledge state and knowledge update.

The framework presented here provides a semantic description of tasks in terms of epistemic change. We take a "model-based" approach in our characterization of an agent’s knowledge about the solution of a problem. Model-based approaches to reasoning have been proposed by several people [49, 51, 53].

A knowledge base (KB) is here presented in terms of models. In its most elementary form we think of a KB as the set of all models under which it is true. Computationally this set is usually too large. However, computational inviability is not inherent to a model-based approach. Research has been directed to keep the set of models of a KB as small as possible. We refer to Khardon and Roth [53] for a detailed description of this. Their approach will be briefly described in section 6.5.1.

6.2 The dynamics of tasks

The framework presented here will be model-based in the sense that the search for a solution will somehow be linked to finding one or several models which pass some epistemic test. This epistemic test is the criterion of the problem, which describes when a solution has been found.

The knowledge of an agent is related to the possibilities it considers at a given moment in time. Problem solving can be described in terms of increasing knowledge about candidate solutions. The amount of knowledge an agent has about the solution is inversely proportional to the number of candidate solutions it considers. As the number of possible solutions decreases the amount of knowledge about solutions increases. In this light problem solving is a process of acquiring knowledge about possible solutions. This means that one could ask at any stage during the problem solving process of an agent: what does the agent know about the solution of the problem? The answer will be a characterization of which solutions will be possible and which are not.

6.2.1 Problem solving in terms of changing knowledge

An example taken from van Benthem [91] may clarify what kind of problem solving we have in mind.

Example 6.1: Consider a game of Master Mind where the purpose of the game is to guess the positions of coloured pegs.

<table>
<thead>
<tr>
<th>Guess</th>
<th>Answer</th>
<th>Open options</th>
</tr>
</thead>
<tbody>
<tr>
<td>START</td>
<td></td>
<td>24</td>
</tr>
<tr>
<td>red, orange, white</td>
<td>● ●</td>
<td>6</td>
</tr>
<tr>
<td>white, orange, blue</td>
<td>● ●</td>
<td>2</td>
</tr>
<tr>
<td>blue, orange, red</td>
<td>● ● ● ● ●</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 6.1
A game of Master Mind.

In each round the player is allowed one guess, after which he gets feedback by way of a number of open or closed dots. An open dot indicates a right colour at a wrong position, a
closed one a right colour at a right position. Table 6.1 shows a game with three pegs and 4 colours (red, white, blue and orange). The arrangement to be guessed is (red, white, blue). The column on the right shows the number of open options after each round. This style of reasoning can be described as an interplay of incoming information and changing knowledge about solutions.

A knowledge level description of a problem and the process of solving it, involves a characterization of what the player knows before, during and after having solved the problem. In general we give an account of the problem solving process of a knowledge-level agent, in the sense used by Newell [67].

In case of the Master Mind game, before the start of the problem solving process the player, or agent, knows a number of things: he (assuming the player is male) knows the rules of the game, the possible colours of the pegs etc. Also the player knows that only one configuration can be a solution, and that there is no sense in considering the conjunction of several configurations. All such knowledge will remain invariant during the problem solving process; it is not subject to change during the problem solving process.

The dynamic or variant knowledge the player has is mainly about the solution of the problem he faces. It is safe to say that before problem solving the player has no knowledge about the solution. In other words: before problem solving any configuration of the pegs could be a solution. After having solved the problem this situation is changed and the player knows which configuration is the solution. During problem solving the number of possible configurations has steadily decreased, as is evident from table 6.1. In other words his knowledge about the solution has increased.

This view on knowledge is elaborately explained and refined by Dretske in his book Knowledge and the Flow of Information [23]. In this view knowledge and information are related notions and are strongly linked to the reduction of possible states of affairs. We adapt and use this theory here in order to present a knowledge level account of problem solving.

### 6.2.2 Knowledge and Information: states and updates

We describe problem solving in terms of knowledge in a dynamic way. By this we mean that we give an account of how knowledge changes while the problem is being solved. In order to do this we introduce knowledge states and state transitions. A more formal account of the notions introduced here will be given later.

A knowledge state is intended to capture what an agent knows. However, as we will see, not everything which would qualify as knowledge will be represented in a knowledge state. For example, procedural knowledge will not be part of an agent's knowledge state.

When some sentence is accepted in a certain knowledge state then the agent knows the sentence. By "accepting" we mean that the knowledge state does not contain anything which refutes the sentence. A more formal definition will be given later.

A knowledge state consists of two types of knowledge: the domain knowledge which will remain invariant during problem solving, and knowledge regarding the possible solutions to the problem. Hence a knowledge state will have a static and a dynamic part.

As remarked above, knowledge is strongly linked to possibilities. Logically, possibilities will later be described as models. From this perspective there are two knowledge states which are of special interest. The first one is the state in which everything is possible. This one is called the
**minimal knowledge state** because if everything is possible then nothing is known. The other knowledge state is the one in which nothing is possible, it is empty or logically inconsistent. This state is known as the **absurd state**.

The domain knowledge associated with a problem will be part of any state during the problem process. However, the dynamic part of the state changes. Since we use the dynamic part of the knowledge state to keep track of the possible solutions to the problem, we will refer to it as the **solution state**.

A **state transition** gives an account of how one state may change into another one. This will be done by means of **updates**. We distinguish two types of updates: **reactive or informational** updates (called 'updates' for short), and **proactive** updates. The first ones intend to capture passive state transitions on the basis of incoming information. For example, knowledge in diagnostic problem solving may change after observations have been made. These observations can be seen as containing new information which affects the knowledge of the agent about possible malfunctions of the artefact to be diagnosed. In other words: the knowledge state is updated with new, observed information.

The reason why we call such updates 'passive' is that we will only deal with expanding knowledge states, not with revisions. Any knew knowledge state which is the result of a state transition will contain at least as much knowledge as its predecessor. (In terms of possibilities the states contract rather than expand, since increase of knowledge means decrease of possibilities.) In the belief revision literature [34] belief state revisions are described when incoming information is inconsistent with beliefs held by the agent. In this case some beliefs should be given up or adapted to conserve consistency of the belief state.

Our framework will be limited in that we only deal with states of expanding knowledge and not with revisions. When information is inconsistent with the knowledge of a state this will lead to the transition to the absurd state. Informational updates are passive because they give rise to purely deterministic changes of knowledge states.

We also make use of **proactive** updates. These updates reflect the rational capabilities of an agent, they often involve the selection of a certain element, a guess or a move. Proactive updates are state transitions which are initiated by the agent itself, rather than a deterministic reaction to information from the environment.

Proactive updates are indeterministic and therefore it is often necessary to return to previous states. In other words: the agent needs some kind of memory to backtrack on the choices it made. Like informational updates we restrict proactive updates by only allowing that a state will have as least as much knowledge as its predecessor.

In the Master Mind game example the guess of the agent at each round can be represented as a proactive update. By guessing a configuration the player ignores all other candidate solutions and changes to a solution state where only this guess is a solution. It's important to note that this move is done on the initiative of the agent itself, and is therefore proactive. The indeterministic aspect of proactive updates is given by the fact that a guess can be right or wrong. When the guess is wrong the agent can, by using it's memory, move back to his previous state. When his guess is affirmed to be correct a solution has been reached.
6.2.3 Epistemic test and criteria

Similar to updates are epistemic tests [95]. Epistemic tests (or 'tests' for short) do not lead to state transition. Instead they are hypothetical updates which are used to test which effect an update would have on the current knowledge state.

As an example (which will be treated in detail later) consider the question whether a given expression is consistent with a certain knowledge state. The answer to this question is either 'yes' or 'no' and there is no need for a transition to a new state. This test for consistency is an example of an epistemic test.

Another example is the test for acceptance of some expression with respect to a given state. (We will present a formal meaning of this notion later.) By this we mean the question whether the expression can be derived from the given knowledge state. Here again we get a boolean answer (yes/no) as output of the epistemic test.

An important use of epistemic tests in our framework is as a specification of a final state, i.e. a state wherein a solution to the problem has been reached. This is done by defining the criterion of the problem as an epistemic test. As mentioned in chapter 4 the criterion of the problem is an expression which states when a candidate solution is in fact a solution.

In the case of the Master Mind game the criterion is that a solution should match the target configuration. This can be formulated as an epistemic test as follows: The test succeeds for a given solution state if it contains only one configuration which matches the target configuration. Otherwise it fails.

Epistemic tests are also used here as boolean constructs in the description of control flow. Problem solving is described as a series of tests and updates together with some flow of control. The flow of control describes the order of updates and test together with possible iterations.

As an example we will give a description of the Master Mind game in terms of a diagram. In figure 6.2 a pictorial representation is given in a flow-chart notation. States are depicted as rectangles, updates and epistemic tests as ovals. The tests are followed by a choice symbol. The diagram should be read from left to right, top to bottom, which can be emphasized by numbered arrows.

The player begins with an initial knowledge state in which all configurations of pegs are possible. We have already remarked that the player will only consider single solutions. Hence the solution state which contains all possible single configurations will be the initial state of a Master Mind player. The player then starts with guessing a configuration. This is modelled as a proactive update, as was explained above. We have not indicated how the player comes to his guess.

The result of the update is a solution state in which only this guessed solution is present. The other candidate configurations are, at least for the moment, disregarded and the transition to a solution state with only the guessed configuration has taken place. The proactive update is recorded into memory for backtracking purposes. This is not shown in the figure. Next, it is tested whether the new state, containing the guessed configuration, meets the criterion. The result of the test is a boolean value and a choice what to do next for each value is given. If the test for this guessed solution succeeds this selection embodies his knowledge about the solution. The problem is solved.

If the test fails the agent has to return to all other candidate configurations. The memory associated with proactive updates, is used to return to the previous state. In addition, the negative feedback has resulted in some new information about the solution. First, the player received the
information that the guessed configuration can safely be discarded. Second, the open and closed dots contain information about which colours should, or should not be at certain positions. Hence, other possible solutions can be discarded as well. This line of reasoning is typical of informational updates. The solution state contracts after processing the information given by the feedback, in the sense that it contains less possible solutions. It is then tested whether this new solution state is not empty. If it is and contains no possible solutions then the procedure stops.

If there are still possible solutions to be considered then the procedure repeats itself until a solution has been found. Hence, knowledge states move progressively to more knowledge about solutions and only backtracking to previous states is provided for proactive updates.

The general line of reasoning in this example is a select-and-test method augmented with a capacity for pruning. The feedback the player receives enables him to disregard candidate solutions. If we keep this feedback to a minimum and only inform the player whether his guess was right or wrong (without given him information in the form of open and closed dots) the result would be a select-and-test method Figure 6.2 can therefore be seen as a general knowledge-level description of select-and-test. Note, that the figure is an informal, high-level description of problem solving and is not intended to be complete.
6.3 Tasks and models

The approach we take in this chapter associates a solution to a problem with one or more models. This "model-based" approach is intuitively very suited for tasks in which a set of candidate solutions is given at the start of problem solving. Other problems, like those of tasks in which the solution should be constructed rather than selected from a predefined list, are more difficult to represent.

The distinction between selecting and constructing solutions is reminiscent, but not identical, to the distinction between analytic and synthetic tasks [82].

- **Analytic tasks** are those tasks in which the artefact or "system" one reasons about is given and the goal of the task is to identify the solution(s) which meet some criterion. Examples of analytic tasks are classification and diagnosis.

- In **synthetic tasks** a "system" or artefact has to be constructed from more primitive elements. Planning and assignment are examples of synthetic tasks. In planning one constructs a plan, in assignment problems an assignment.

In analytic tasks like classification and diagnosis one is given a set of candidate solutions, from which one (or several) must be selected which meets a given criterion. In synthetic tasks like planning one constructs a plan as a series of actions to reach a given goal. However, it is sometimes possible (and even feasible) to construct all possible solutions first, and then to select the best "constructed" solution. Hence a constructive problem can sometimes be solved by methods typical for analytical tasks.

In the framework we describe the solution is associated with one or more models, and states represent which possible solutions an agent considers at a given time. Incoming information and pro-active behaviour induce state change and eliminate possibilities. Finally the agent reaches a state in which a criterion is met, and the problem has been solved. This final knowledge state then contains a solution to the problem.

Intuitively, this approach suits tasks, in which the solution is primitive, very well. We will give an example of how several classification criteria can be described later in this chapter. Such an example for synthetic tasks will not be given here. A brief description of an update system for grouping problems is given in chapter 8.

However, model-based approaches have been used for planning. As an example we mention Kautz et al. [52]. They describe a formal model for planning problems based on the notion of satisfiability rather than deduction. As a consequence they identify a solution to a planning problem (a plan) with a model. In addition the representation of domain knowledge is done in such a way that any model of the domain theory corresponds to a given plan. This is similar to the approach we take here and a description of planning in our framework would closely follow the one given by Kautz et al..

Our approach can be seen as an extension or generalisation of such a model-based approach. The differences are two-fold. First, we use update semantics as the underlying formalism. This allows for a dynamic description of the model-finding process. We also make use of the distinction between passive and pro-active updates to characterize state changes. This enables us to characterize state change based on the nature of the information an agent receives.

Second, our representation is not limited to sets of models. As we will see, states can be structured in many ways, and hence one can make use of preferential, or minimal reasoning. As
a consequence our approach is not limited to satisfiability. In fact, satisfiability is one example of the more general notion of epistemic test.

In addition to these formal differences, our motivation also differs from those of Kautz et al. One of the main motivations for the "planning-as-satisfiability approach" is to be able to construct fast and efficient methods for planning problems. Here, however we look merely at the functionality of methods, and their description in terms of knowledge-level constructs. Having said that, the results of computational model-based approaches, as described by Kautz et al. [51] and Khardon et al. [53], can be used to make our approach computationally viable. We will describe this after having explained the underlying formalism of update semantics.

### 6.4 Logical Dynamics

We will proceed by formalizing the ideas explained above. We do this by first explaining an update system described by Veltman [95]. The main idea behind this system is that it gives a semantic and dynamic account of reasoning with states and updates. We will extend and adapt this system later for our purposes of describing knowledge level tasks, and now proceed with the treatment of the logical aspects.

In logic the dynamics of information flow has attracted considerable attention. Various formalisms have been developed which can be characterized as evolving around the notion of information rather than that of truth. For an overview and discussion of this subject we refer to van Benthem [91].

The central idea behind update semantics [95] is the following. Whereas according to the classical view, the meaning of a sentence is given by the conditions under which it is true, in the dynamic account the meaning of a sentence is given by the change it induces to an information state.

This view on meaning is more attractive from a 'cognitive' perspective. One can think of an agent, possessing a state, dealing with incoming information and making changes accordingly. However, from a formal point of view this change of perspective may not necessarily offer something new. Classical logic can be given a dynamic interpretation quite easily, as will now be shown.

#### 6.4.1 Propositional logic dynamified

Consider a propositional language \( L \) with propositional letters \( p, q, r \ldots \) and the usual connectives. We associate with \( L \) the powerset \( W \) of atomic sentences. An information state \( \sigma \) is defined as follows:

**Definition 6.1**: Let \( W \) be the powerset of the set \( A \) of atomic sentences in a language \( L \). An information state \( \sigma \) is any subset of \( W \). In symbols: \( \sigma \subseteq \wp(A) \). The elements of a state, being sets of atomic formulas, are called (possible) worlds.

The definition of an information state may change when one describes richer update systems. The notion of knowledge state will be used as a higher level concept. We use it to express the knowledge an agent has at a given moment in time. The notion of information state will be used in a more rigorous, logical sense.

---

1Readers familiar with Veltman's work can skip all but the latest subsection of this section.
If $\sigma$ is a state and $\Phi$ a sentence we write $'\sigma[\Phi]'$ to denote the update of $\sigma$ with $\Phi$. Here $[ ]$ is a function which assigns to each sentence $\Phi$ an update operation $[\Phi]$.

For each sentence the update function can be defined as follows:\(^2\):

- **atoms**: $\sigma[p] = \sigma \cap \{ w \in W | p \in W \}$
- **\neg**: $\sigma[\neg \Phi] = \sigma \cup \sigma[\Phi]$
- **\wedge**: $\sigma[\Phi \wedge \Psi] = \sigma[\Phi] \cap \sigma[\Psi]$
- **\lor**: $\sigma[\Phi \lor \Psi] = \sigma[\Phi] \cup \sigma[\Psi]$

A sentence $\Phi$ is **acceptable** in state $\sigma$ iff $\sigma[\Phi] \neq \emptyset$. A sentence $\Phi$ is **accepted** in state $\sigma$ iff $\sigma[\Phi] = \sigma$. If $\Phi$ is accepted by $\sigma$ we write $\sigma \vdash \Phi$. When $\Phi$ is accepted in every state, we write $\vdash \Phi$. An update is **informative** if $\sigma[\sigma] \neq \sigma$.

It is not difficult to discern the classical semantics of propositional logic in this formalism. A state is a set, whose elements, or worlds, are sets of atomic propositions. Each world corresponds to a valuation in the following way: atomic propositions occurring in the world are given the truth value 'true', those which are absent 'false'. Hence, a state can be seen as a collection of valuations or models, and $W$ as the collection of all models of $L$. $\wp(A)$ is denoted by $\emptyset$ and is called the **minimal state**. $\emptyset$ is called the **absurd state** and is denoted by $1$. Note that all formulas are accepted in the absurd state, hence the name. Logically, $\emptyset$ and $1$ can be thought of as the symbols $\top$ and $\bot$ respectively.

**Example 6.2**: Consider a language $L$ with three propositional atoms $p, q, r$. The powerset $W = \wp(\{p, q, r\})$ is given by $\{\{p, q, r\}, \{p, q\}, \{p, r\}, \{q, r\}, \{p\}, \{q\}, \{r\}, \emptyset\}$. The elements of the set are called worlds, which can be interpreted as models. The set $W$ is itself an information state, called the **minimal state** $\emptyset$, containing all models over $L$.

**Updating the minimal state**: $\emptyset = \wp(\{p, q, r\})$ with $p$ gives us a new state which we call $\sigma$:

\[
\sigma = \emptyset[p] = \{\{p, q, r\}, \{p, q\}, \{p, r\}, \{p\} \}
\]

In this new state $p$ is accepted ($\sigma \vdash p$) because $p$ occurs in every world of $\sigma$.

If we start with the powerset $W$ as our initial state than updating sentences to this state is equivalent to adding them to a set of hypotheses. So $W[p] \vdash p$ is the same as $p \vdash p$ in classical propositional logic. And the truth of a sentence $\sigma$ is equivalent to $\sigma$ being accepted in $W$. In this sense the dynamic variant does not offer anything extra compared to the classical, static, truth-functional logic.

Instead of $\emptyset[\sigma]$ we will sometimes write $\|\sigma\|$. Propositional sentences can now be given a static meaning as follows:

\[
\|p\| = \{w \in W | p \in w\}
\]

\[
\|\neg \sigma\| = W - \|\sigma\|
\]

\[
\|\sigma \wedge \psi\| = \|\sigma\| \cap \|\psi\|
\]

\[
\|\sigma \lor \psi\| = \|\sigma\| \cup \|\psi\|
\]

We will say that $\|\sigma\|$ is the set of worlds in which the proposition $\sigma$ holds. When one thinks of worlds as models (truth assignments of the atomic formula), $\|\sigma\|$ is the set of all models which

\(^2\) $- -$ denotes the difference between the sets.
make $\phi$ true. For example, the static meaning of an atomic formula $p$ (written as $\|p\|$) is equal to the set of worlds in which $p$ occurs.

Hence, such an update system is no different from a truth-functional propositional logic. However, this parallel between dynamic semantics in terms of states and updates with static, truth-functional semantics changes when the dynamic language is extended. The framework of update semantics facilitates the definition of other operators. For example the language can be enriched with an operator for an 'epistemic test'.

### 6.4.2 Epistemic tests

The propositional system described above can be extended by introducing an epistemic test. As we remarked above, unlike updates, tests do not lead to state change. Instead updates are done hypothetically to see if the desired result would occur. In fact, acceptance $(\models \top)$ is an epistemic test as well.

We enrich the system described above with a test for consistency. The propositional language is extended with an operator $\Diamond$, \textsuperscript{3} so that propositions like $\Diamond \phi$ can be formed. Informally its meaning is as follows. If an update with $\phi$ to the state $\sigma$ does not lead to the absurd state $1$ then the test $\Diamond \phi$ succeeds, otherwise it fails. Its dynamic semantics can be defined as follows:

$$
\Diamond: \quad \sigma[\Diamond \phi] = \sigma \text{ if } \sigma[\phi] \neq \emptyset \\
\sigma[\Diamond \phi] = \emptyset \text{ otherwise }
$$

Hence the epistemic test $\Diamond \phi$ consists of the hypothetical update of $\phi$ to a state $\sigma$. If the result is absurd, then in truth-functional terms $\phi$ is inconsistent with the information contained in $\sigma$, otherwise it is consistent with it.

An alternative formulation would be:

$$
\Diamond: \quad \sigma[\Diamond \phi] = \sigma \text{ if there is a non-empty state } \sigma' \text{ such that } \sigma' \subseteq \sigma \text{ and } \sigma[\phi] = \sigma' \\
\sigma[\Diamond \phi] = \emptyset \text{ otherwise }
$$

From this it follows that a sentence of the form $\Diamond \phi$ is accepted in a knowledge state $\sigma$, $\sigma \models \Diamond \phi$, iff its update with $\phi$ would not result in the absurd state $1$.

It is clear that by introducing $\Diamond$ the parallel between acceptance and the classical notion of truth is disturbed. A sentence like $\Diamond p$ can be accepted but will not be true in the classical sense. To see this, consider a state where $\Diamond p$ is accepted, then update it with $\neg p$. In the resulting state $\Diamond p$ is no longer accepted. Hence acceptance is no longer monotonic.

Not mentioned by Veltman is the fact that adding this epistemic test for consistency to a propositional update system, results in a system which is equivalent to the modal logic S5. This logic is often used for knowledge representation. For example, in Fagin et al. [24] it is used to represent problem solving in a multi-agent setting.

S5 is defined as the modal system in which all propositional tautologies hold together with the following axioms:

- $\Box(\phi \to \psi) \to (\Box \phi \to \Box \psi)$
- $\Box \phi \to \phi$

\textsuperscript{3}Veltman uses might instead of $\Diamond$. 

---

**A dynamic account of knowledge level tasks**

73
6.4.3 Preferential reasoning

The update system presented here can be further extended. Various non-classical ways of reasoning can be modelled. Veltman in [95] describes a default system in terms of the operators normally and presumably. Adding such operators also involves a change to the definition of information state.

The idea of incorporating defaults can be put as follows. Sentences like normally $\phi$ can be used to express what normally is the case. In [95] they are called default rules. Suppose that an agent has a set of sentences $P$ of default rules. Every world in the agents knowledge state could be a model for some of these sentences. Worlds in a state $\sigma$ can be ordered with respect to the set $P$ as follows. We describe the order $\leq$ between worlds $v$ and $w$ from $\sigma$ as follows: $w \leq v$ if $w$ is a model for all the sentences from $P$ which hold in $v$ and maybe some more.

To explain this in other words: The set $P$ contains sentences which describe what the agent knows normally to be the case. A world (seen as a truth-functional model) in which every sentence in $P$ holds represents the situation in which everything is normal to the agent. The world in which no sentence of $P$ holds represents the situation in which nothing is normal.

Each world can be seen as a description of a possible state of affairs. Some worlds are more normal than others and hence worlds can be ordered along the relation 'more-normal-than' with respect to the set $P$ of sentences which are to be regarded normally to be the case. A state then becomes an ordered set of worlds.

This means that the notion of information state is extended with a partial ordering on worlds. This ordering can then be used as a preference structure. Updates with sentences of the form normally $\phi$ to a state $\sigma$ affect the ordering of the worlds in $\sigma$. A new epistemic test in the form of presumably $\phi$ can now be defined to see whether $\phi$ holds in the most preferred, or optimal world, being the one highest in the ordering. Other updates work exactly as before and may eliminate worlds, leaving the remaining ordering intact.

Such an example shows that the notion of information state can be changed in the framework of update semantics to allow the description of preference structures and non-monotonic reasoning. Furthermore it allows the description of changing preference structures during reasoning.
6.4.4 Formalizing defaults

Consider a propositional language $L$ which we extend to $L'$ with operators $n$ and $p$ such that if $\phi$ is a formula in $L$ then $n(\phi)$ and $p(\phi)$ are formulas in $L'$.\footnote{$p$ can be read as presumably, $n$ as normally, as Veltman does. However, we prefer this more abstract notation. A more abstract reading of these operators is to view $n$ as a default rule. $p$ can be read as the test whether the default holds in a state.}

By $W$ we denote the powerset of the set of atomic propositions $A$. The elements of $W$ are denoted by $w, v \ldots$ and are called worlds. An expectation pattern $\varepsilon = (W, \leq)$ is a partially ordered set (poset) on $W$. If $w \leq v$ and $v \leq w$ we write $w \equiv v$, for worlds $w, v$.

The use of expectation patterns is as follows. The agent has some knowledge of state of affairs which it expects normally to be the case. This is represented in the form of sentences of the form $n(\phi_1) \ldots n(\phi_n)$. The set of all sentences which the agent considers normally to be the case is denoted by $P$. $\langle w, v \rangle$ is an element of the expectation pattern $\varepsilon$ iff every proposition in $P$ which holds in $v$, also holds in $w$. (We say that the sentence $\phi$ holds in world $w$ if $w \in \mathbb{I} \phi \mathbb{I}$. That is, if $w$ can be interpreted as a model for $\phi$.)

Hence all the worlds are partially ordered relative to the set $P$ of sentences which are normally to be the case. The world $w$ such that $w \leq v$ for all $v \in W$, is called a normal world. Hence, $w$ is a world in which everything considered normally to be the case, does actually hold. In the following we assume that at least one such a world exists for states. In truth-functional terms this means that the set of sentences that express what is normal, is consistent.

An information state for an update system with default rules can now be defined as a pair $\sigma = (\varepsilon, s)$ where $\varepsilon$ is a pattern on $W$ and $s$ captures the knowledge of the agent, $s \subseteq W$. The minimal state is given by $\langle W \times W, W \rangle$, the absurd state by $\langle \{ \langle w, w \rangle | w \in W \}, \varnothing \rangle$.

As an example of how this all works consider figure 6.3. The figure contains a graph of a state $\sigma = (\varepsilon, s)$ for a language of three atoms. The eight worlds are denoted by $w_0 \ldots w_7$.

Figure 6.3
A knowledge state with eight ordered worlds. If $w < v$ then there is a path to the left from world $w$ to world $v$. If $w \equiv v$ $w$ and $v$ are placed in the same oval (taken from Veltman [95]).

Worlds are placed in the same oval if they belong to the same equivalence class defined by $\equiv$. If for worlds $w, v$ we have $w < v$, then there is a rightward path from the oval containing $w$ to the one containing $v$. In such a case $w$ is more normal than $v$.\footnote{\varepsilon can be read as presumably, $n$ as normally, as Veltman does. However, we prefer this more abstract notation. A more abstract reading of these operators is to view $n$ as a default rule. $p$ can be read as the test whether the default holds in a state.}
In this situation depicted in figure 6.3 there are two normal worlds: \( w_0 \) and \( w_5 \). The set \( s \) captures the knowledge the agent has about the world. It is pictured as a triangle and contains \( w_6, w_3 \) and \( w_4 \). Hence, the agent knows that the other worlds are no longer possible. Since both normal worlds \( w_5 \) and \( w_0 \) do not belong to \( s \), the agent knows that not everything is normal. However it may have a preference for worlds in \( s \) that are 'as normal as possible'.

Such worlds are called optimal in the following sense: \( w \) is optimal in \((\varepsilon, s)\) iff \( w \in s \) and there is no \( \nu \in s \) such that \( \nu < w \). In figure 6.3 both \( w_3 \) and \( w_6 \) are optimal in the pictured state.

Optimal states play an important role when knowledge is incomplete. The expectation pattern orders worlds along the agent's knowledge of what is normal. The set \( s \) captures the knowledge of what is, and what is not, possible. An optimal world then captures the knowledge of what is to be expected given the knowledge of what is still possible. Optimal worlds will come into play when one considers sentences of the form \( p(\phi) \). Such a sentence is accepted if \( \phi \) is true in all optimal worlds.

Three kinds of updates (or tests) can now be distinguished.

- Propositional updates: updates of sentences not containing \( p \) or \( n \). These work exactly as in the propositional update system, and affect \( s \), while leaving the pattern \( \varepsilon \) untouched.

- If \( \phi \) is of the form \( p(\psi) \) then \( \sigma[\phi] \) is given as:

\[ \text{Let } m = \text{the set of optimal worlds in } \sigma, \text{ then } \sigma[\phi] = \sigma \text{ iff } \sigma[\psi] = m. \text{ Otherwise } \sigma[\phi] = 1. \]

In words: \( p(\psi) \) is accepted in \( \sigma \) if and only if \( \psi \) holds at all optimal worlds in \( \sigma \).

- If \( \phi \) is of the form \( n(\psi) \) then an update will result in a change of the pattern \( \varepsilon \), on the condition that there is at least one normal world in which \( \psi \) is true. If there is no such normal world \( \sigma[\phi] = 1 \).

Otherwise the pattern \( \varepsilon \) is refined. A pattern \( \varepsilon' \) is a refinement of \( \varepsilon \) if \( \varepsilon' \subseteq \varepsilon \). And a pattern \( \varepsilon \) is a *refinement with the proposition \( e \)*, \( \varepsilon \circ \varepsilon' \) if \( \varepsilon \circ \varepsilon = \{ (w, \psi) \text{ if } w \in \varepsilon \text{ then } \nu \in \varepsilon.} \)

The update of \( \sigma[n(\psi')] \) can now be defined as being equal to \( \langle \varepsilon \circ ||\psi'||, s \rangle \).

Note that an update with a sentence of the form \( p(\phi) \) is actually an epistemic test: it does not change the information state.

As an example of how this update system works, consider figure 6.4. Here four worlds are represented. \( W = \{ w_0, w_1, w_2, w_3 \} \), and \( w_0 = \{ p \}, w_1 = \{ q \}, \) and \( w_3 = \{ p, q \} \).

The figure starts with the minimal state given by \( \langle W \times W, W \rangle \). Hence at this point the agent has no knowledge of what is normally the case. Then this state is updated with \( n p \). This induces an ordering, and puts worlds containing \( p \) to the left of those which do not contain it. Notice that the knowledge of which worlds the agent considers possible, given by \( s \), has not changed by this update.

It is quite easy to establish that \( p(p) \) holds in this new state. Updating this state with the proposition \( q \) eliminates two worlds (\( w_0 \) and \( w_1 \)) from \( s \). \( w_3 \) is both optimal and normal in this state. It can be readily verified that in addition to \( p(p), p(q) \) now also holds.

An alternative to the last update is shown by the vertical arrow. Here the second state is updated with \( nq \), affecting the ordering of the worlds. \( w_3 \) has become the only normal world in

\footnote{Here the proposition \( e \) is the set of worlds in which the sentence expressed by the proposition holds. So if \( \phi \) is the sentence, \( ||\phi|| \) is the proposition expressed by the sentence.}
An example series of updates concerning four worlds, depicted by numbers. Patterns are pictured similar to the previous figure. The set $s$ of possible worlds is shown as a dashed rectangle. Updates are shown as arrows with their respective formulas.

this state. It is eliminated from $s$ by the update of $\neg(p \land q)$. Hence in the new state $w_1$ and $w_2$ are both optimal.

6.4.5 Another epistemic test

The above preferential system was described by Veltman in [95]. We add a new epistemic test to it which will be used later on in the description of classification criteria.

First we look at the test $p\phi$ which we used above. It can be defined as follows:

Let $W_o$ be the set of all the optimal worlds in a state $\sigma$.

\[
p: \quad \sigma[p(\phi)] = \sigma \text{ iff } W_o \cap \|\phi\| = W_o.
\]

Otherwise $\sigma[p(\phi)] = \emptyset$.

In words: $p(\phi)$ holds in $\sigma$ if it holds in the state formed by all optimal worlds in $\sigma$. Veltman uses this as to express presumably. A sentence presumably $\phi$ is accepted in a state $\sigma$ when it holds in all those worlds which are considered normal in $\sigma$.

It is easy to think of a weaker version of this test. It is a test for consistency in the state formed by all optimal worlds in $\sigma$.

Here follows its definition:

\[
\Diamond p: \quad \sigma[\Diamond p(\phi)] = \sigma \text{ iff } W_o \cap \|\phi\| \neq \emptyset.
\]

Otherwise $\sigma[\Diamond p(\phi)] = \emptyset$.

Note that $\Diamond p(\phi)$ can also be defined as $\neg p(\neg \phi)$. In order to make use of this operator the language $L'$ has to be extended to allow sentences of the form $\Diamond p(\phi)$.
6.5 Representing knowledge dynamically

The idea of an information state as it occurs in update semantics will here be used to capture the knowledge an agent has at a given moment. When used in this sense we will use the term knowledge state. If an agent starts with the minimal state 0 as its initial state, this represents the fact that the agent has no knowledge at all, (hence the notion of minimal state). After successive updates, the knowledge of the agent will increase as its state will contract. The state will expand in terms of knowledge. The amount of knowledge and worlds in a state are each others duals.

6.5.1 Models, methods and tableaux

Before we apply the logics to our framework we would like to emphasize the semantic nature of this exploration. Update semantics can not be seen as a computational viable method for reasoning. Building a propositional theorem prover which begins with a powerset of all atoms is not a good idea.

The analysis presented here is not aimed to be a computational analysis, targeted at the development of fast or new PSMs. Instead we provide a functional, knowledge-level, rather than a computational efficient, symbol-level description of task reasoning. We claim that knowledge is much more a semantic than a syntactic notion and therefore a semantic account is worthwhile.

However, there are ways of turning the framework presented here into a computational effective way of solving problems. One approach is to represent information states not as sets of all models (worlds) but to limit this set to a number of characteristic models. This is described in detail by Kautz et. al [51] and Khardon et. al [53]. Kautz et al. describe their approach for propositional Horn theories. Briefly, the idea is as follows: given the set of all models of some theory \( \Sigma \), this set can be represented by a subset of models, called the characteristic models of \( \Sigma \). These characteristic models can be seen as a minimal "basis" for all models of \( \Sigma \), in the sense that one can generate all models by taking the conjunctive closure of the set of characteristic models. (For details we refer to Kautz et. al. [51].) As a result a model-based approach to knowledge intensive reasoning becomes computationally very efficient. For example, in the same paper, Kautz et. al. show that abduction can be performed in polynomial time.

Characteristic models can be used in the representation of information states of update semantics. Instead of defining a state as a set of worlds one uses only the characteristic worlds of that state. How to work out all details, particularly for the case of ordered worlds and default rules, remains a interesting topic for further research.

Another approach of making update semantics a computationally efficient way of describing problem solving is to relate it to theorem proving techniques. An interesting option is to use analytic (or "semantic") tableaux. Tableaux have been used in chapter 3. They are particulary interesting because of their 'semantic properties'. Except being used as a classical proof method, tableaux can be used for other purposes, like model finding, minimal entailment [68] and abductive reasoning [62, 3]. Tableaux have the nice property that they combine syntactical and semantic ideas. They can be considered as purely syntactical structures with rules for their manipulation. On the other hand they can be interpreted semantically in the sense that each open branch corresponds to a model of the represented theory.

The propositional update system with tests for consistency and acceptance can very well be translated into a computational more attractive format. We will show how to use analytical
tableaux for such an update system.

There is a straightforward relation between tableaux and the (non-preferential) propositional information states introduced so far. Let $\tau(\phi)$ be the tableau for the formula $\phi$. Two tableaux $\tau_1$ and $\tau_2$ can be joined into a tree $\tau_3$ by adding the root of one of the two trees to all the branches of the other. We use the notation $\tau_1 + \tau_2 = \tau_3$ for the operation where the root of $\tau_2$ is added to the branches of $\tau_1$. We denote the tableau of a knowledge state $\sigma$ by $\Upsilon(\sigma)$. Its meaning is as follows:

- The minimal state: $\Upsilon(W) = \tau(\top)$ (The empty tree).
- The absurd state: $\Upsilon(\emptyset) = \tau(\bot)$ (Any closed tree).
- Update: $\Upsilon(\sigma[\phi]) = \Upsilon(\sigma) + \tau(\phi)$.
- Consistency test: $\Upsilon(\sigma[\phi])$:
  - If $\Upsilon(\sigma) + \tau(\phi)$ closes then $\Upsilon(\sigma[\phi]) = \tau(\bot)$, else $\Upsilon(\sigma[\phi]) = \Upsilon(\sigma)$.
- Acceptance: $\sigma \models \phi$: The tableaux $\Upsilon(\sigma) + \tau(\neg \phi)$ closes.

The minimal state corresponds to the tableau for $T$: all possibilities are open and no branches have been closed. The absurd state corresponds to the absence of possibilities, and hence all branches have been closed.

The tableau of an update $\sigma[\phi]$ is defined recursively. First construct the tableau for $\sigma$ which can be seen as series of updates of the minimal state. Then add the root of the tableau for $\phi$.

The consistency and acceptance tests are pretty straightforward. To test a formula for consistency with a theory one checks whether adding the tableau of the formula to the theory does not lead to closure of the resulting tableau. The same is done in terms of states. Acceptance tests whether a formula follows from the theory/state. One adds the negation of the formula to the tableau of the the theory/state and checks for closure.

In order to get from a tableau $\Upsilon$ of a theory $\Sigma$ to an information state $\sigma$ we make use if the fact that each open branch of $\Upsilon$ represents a model. More specifically an open branch corresponds to one or more worlds. Note that a tableau of a theory only contains those atoms which occur in the theory, whereas worlds refer to all atoms in the language. Therefore there can be more than one world corresponding to an open branch.

To find the worlds corresponding to a branch we look at the positive and negative literals. 6 We then select all worlds which contain all positive literals and do not contain the duals of the negative literals. If we repeat this for every branch we end up with the set of worlds which forms the corresponding state.

**Example 6.3:** Consider a language with three propositional atoms $p, q, r$. The minimal knowledge state is then formed by taking the powerset $\varphi(\{p, q, r\})$. The tableau of this state $\Upsilon(\varphi(\{p, q, r\}))$ is the tableau $T$. Updating the minimal state with $\neg r$ removes from the powerset all sets in which $r$ occurs. The associated tableau of this state is the tableau of $\neg r$ (which is just the leaf $\neg r$) added to $T$ which results in the one leaf tableau $\neg r$.

Checking consistency of $r$ in terms of tableaux is done by adding $r$ to the tableau just built and checking for open branches. In this case consistency obviously fails, since the new tableau

---

6A literal is an atom or its negation. Positive literals are atoms, negative literals their negations. The *dual* of a negative literal is the atom without negation.
closes. Acceptance of \( \neg r \) succeeds since adding \( r \) to the tableau of \( \neg r \) results in closure.

Notice that where reasoning with knowledge states involves the reduction of possibilities after updates, tableaux do just the opposite. One starts with an empty tableau and expands it as updates are being made. This has the obvious advantage that not all models are being represented, whereas knowledge states contain all possible models explicitly.

We will not present tableaux for update systems with preferential operators like \( p \) and \( n \). We will leave this subject for further study. We note that tableaux are frequently used in non-monotonic reasoning and minimal entailment [68].

### 6.5.2 Knowledge and solution states

We will use, extend and adapt the above formalism to describe knowledge level problem solving. In update semantics an information state was defined as a subset of the powerset of the atoms of the language. We will use the information states of update semantics as a representation of our knowledge states. However, we add some structure to knowledge states which reflects different types of knowledge.

As described above, the knowledge an agent has can be divided into two categories: a static part and a dynamic part. The static knowledge remains invariant during problem solving, while the dynamic part is subject to change. Hence, when we talk about knowledge states we are mostly concerned with the dynamic part, taking the static part for granted.

In the representation of tasks, domain knowledge, common-sense knowledge and some assumptions do usually make up the static part of a knowledge state. When new information becomes available the agent will be able to make new deductions. Of course, these new deductions are made possible by applying the static knowledge to the new information.

Dynamic knowledge is subject to change. However, not every dynamic sentence of a knowledge state is relevant to the problem solving process. As new information comes in, many useless sentences can be derived. All these qualify in principle as dynamic knowledge.

This problem of which information is relevant can be solved pragmatically. Only knowledge that is related to the possible solutions of the problem is important. We therefore often equate dynamic knowledge with the solution space.

To see how this affects the representation of knowledge states consider the domain knowledge of some problem, formulated in a language. The minimal information state of this language is formed by taking the powerset of the atoms of the language. Next we update this state with every sentence representing domain knowledge. The resulting state then contains the domain knowledge.

From a semantic point of view the worlds of this state represents all the models of the domain knowledge. Since the domain knowledge will remain invariant a model of the solution to the problem will be among the worlds in this state.

Since we are interested mainly in knowledge about possible solutions we want to be able to talk about the candidate solutions which are present in the state which contains the domain knowledge. Therefore we abstract a knowledge state by ignoring all information that is not about possible solutions.

Knowledge states can be abstracted to solution spaces (or solution states as we will call them) in the following way:
**Definition 6.2**: Let $S$ be the set of possible solutions. A solution state $\sigma_s$ is a subset of the powerset of solutions $\wp(S)$.

A solution state $\sigma_s$ is a **representative** of state $\sigma$ iff the following holds:

$$\sigma_s = \{ w_s \mid w_s \subseteq w, \forall w \in \sigma, w_s \in \wp(S) \}.$$  

Hence, a solution state is a representative of a knowledge state $\sigma$ if it can be produced by removing all elements not in $S$ from its worlds. Note that several knowledge states can be represented by a single solution state.

The role of the notion of solution state is to see which possible solutions are considered by the agent with respect to a given state. The abstraction allows us to hide details about knowledge and updates which do not affect the solution space.

One can describe two kinds of updates in terms of solution states. The first are updates with sentences of which the atoms are all candidate solutions. For these it is immaterial whether one updates the underlying knowledge state or the solution state itself, since such sentences will only affect the candidate solutions.

Other updates that can be described in terms of solution states instead of knowledge states are those in which one wishes to hide the details of all changes of knowledge, except those related to the possible solutions. This use is pragmatically motivated. For example, if a knowledge state contains the knowledge that a blackbird is black and we update this state with the observation of a white bird, then a description of the update in terms of solution state will just show the elimination of **blackbird** as a candidate solution. The logical justification of this elimination is of course given by the description of the update to the underlying knowledge state. The solution state description presents a less detailed and more abstract account in which trivial steps do not have to be spelled out. Hence, updates to solution states can be used where one assumes that their effect needs no further elaborate description.

It is not always necessary to think of solution states as subsets of the powerset of all solutions. In many problems one looks for a single solution to a problem. In diagnosis this is called the **single fault assumption** and similarly in classification one usually assumes that a single class should account for the observations. In such cases this information reduces solution sets immediately to singleton solution states (those containing one solution only). We regard such single solution assumption as part of the invariant part of the agents knowledge.

One should be careful how to interpret solution states. Every world of such a state contains zero or more solutions, and can be said to express that it is possible that its elements are solutions. Every world expresses a possible state of affairs. Whether an element really is a solution depends on the criterion of the problem at hand. For example: the criterion of weak explanation expresses that a class is a solution if and only if it occurs in some world. The criterion of strong classification demands that solutions should be necessary, or occur in all worlds of the solution state.

### 6.5.3 Updates

Updates give rise to changes in knowledge states. As we described above, from a knowledge level perspective there are two kind of updates: **reactive** and **proactive** updates.

Formally speaking there is no difference between reactive and proactive updates. The difference lies in the interpretation of what an update means. From a knowledge level perspective reactive updates reflect state changes resulting from processing new information coming from the environment. For example, in diagnosis and classification tasks observations play a key role. The
agent reacts to new information contained in these observations by updating it into his current knowledge state.

In more detail an observation can be represented as a formula \( \varphi \) which is updated to the agent's current knowledge state. Depending on the content of the observation the knowledge of the agent then changes state.

On the other hand, an agent may also make decisions, guesses or explore possibilities within his own knowledge state. These decisions are represented as \textit{proactive} updates which reduce the knowledge state to fewer worlds. As an example consider the Master Mind game where the player makes a guess. We can describe this as a state change of the players solution state in such a way that the new state only contains this guess.

In detail this can be described as follows. Let \( \sigma_s \) be the player's solution state, containing those Master Mind configurations which he judges to be possible solutions. If he makes a guess \( g \) then the result is the new state \( \sigma'_s \) which only contains \( g \) as possible solution. We can describe the proactive selection of one of the possible solutions from a solution state as the result of the update with that choice. Continuing the example, \( \sigma'_s \) is the result of updating \( \sigma_s \) with \( g \).

Hence, "selecting some worlds from a state" and "proactively updating" are two sides of the same coin. To select from a state \( \sigma \) (in a propositional update system) all worlds which contain \( p \), one simply updates \( \sigma \) with \( p \). In this way every selection of worlds from \( \sigma \) corresponds to a sentence which after updating to \( \sigma' \) results in a state containing the desired selection.

A guess is an example of a proactive update. In the Master Mind game it is a selection of one configuration. This selection is proactive because the agent himself takes the initiative for it. However, it should be clear what choices the agent can make and which not. Proactive updates should reflect rational capabilities of an agent which are strongly related to the structure of the knowledge state. In the case of the Master Mind player the choice of \( g \) was a random selection of the solutions contained in \( \sigma_s \). The player then could have chosen another configuration but only among those which were among the candidate solutions in \( \sigma_s \).

A proactive update then reflects a capability of the agent to use the structure of its knowledge state to initiate state change. These capabilities can be described in terms of pre- and postconditions which involve quantifications over epistemic tests.

The precondition describes in terms of epistemic tests whether a proactive update is applicable or not. The postcondition is a description of the state resulting after the proactive update.

Consider a state where some solutions are consistent and others have been ruled out, as in the case of the Master Mind player. In addition assume that solutions exclude each other. Suppose the agent has the capability to proactively update its solution state \( \sigma_s \) to any sub-state \( \sigma'_s \) consisting only of one of these solutions. This capability can be expressed in terms of pre- and postconditions as follows.

\begin{align*}
\text{Precondition} & : \quad \text{There is a solution } s \text{ such that } \Diamond s \text{ holds in the solution state } \sigma_s. \\
\text{Postcondition} & : \quad \sigma'_s = \sigma_s[s].
\end{align*}

Here \( \sigma_s \) is the solution state before the update and \( \sigma'_s \) the solution state immediately after. Another example involves an agent who can produce a solution which is consistent with all optimal worlds of its state \( \sigma_s \).

\footnote{By a substate \( \sigma' \) of a state \( \sigma \) we mean a state which consists of a subset of worlds of \( \sigma \) and preserve the order of \( \sigma \) if there is one.}
Precondition: There is a solution $s$ such that $\Diamond p(s)$ holds in the state $\sigma_s$.

Postcondition: $\sigma'_s = \sigma_s[s]$. There is a choice of how to represent the capabilities of an agent’s knowledge. One could opt for a structured knowledge state with few (or no) proactive updates. Alternatively, one could keep the knowledge state simple and put more emphasis on the definition of the proactive updates.

For example, suppose a Master Mind player when making a guess has a strong preference for making guesses in which red pegs occur. We consider two ways of representing this. First, we could order the worlds in the solution state, giving preference to those containing red-pegged configurations. The proactive update can then be described as the (random) selection of one of the preferred worlds.

Another solution is to keep the knowledge state unstructured and impose no orderings on the different solution. Instead we define the precondition for a proactive update as a test for a solution which contains red pegs. In addition we must also provide a proactive update for the eventuality that there are no more red pegged solutions in the current state. This can be described in the form of a simple random selection of a solution, since the player had no preferences when there are no red-pegged candidate solutions to be chosen.

Another capability that we want to assign to a knowledge-level agent is related to reactive rather than proactive updates. Reactive updates deal with processing incoming information. However, we want to enable the agent to order, to ignore and to select the incoming information. For example, in classification problems many observations are being made. The agent must have the capability to process these observations in a certain order. In addition it must also be able to ignore some observations and select others as more important.

Again, a preferential representation, by means of ordered knowledge states, can be used in such cases. However, there is one principle, which is implicit in the use of information states, which can be used to describe the selection of new information. In general an agent will have a preference for a high information value. A sentence $\phi$ has a higher information value than the sentence $\psi$, relative to an information state $\sigma$, if the update of $\phi$ to $\sigma$ leads to a smaller state than the update of $\psi$ to $\sigma$.

In diagnosis one observation might lead to a bigger reduction of possible solutions than the other and hence has a higher information value. This principle of information value can be seen as an introspective capability of the agent. In chapter 2 it was used to order the observation attributes in rock classification. The system would ask for the value of the attribute with the highest average information value.

Finally, as proactive updates are concerned with making choices, there is always the possibility that a wrong choice will be made. Therefore the agent needs some means of backtracking to previous choice points.

6.5.4 Memory

Proactive updates are strongly linked to a memory. An agent is always allowed to retract its proactive updates. Again the master mind game may serve as an example. When a guess fails to be a solution, the player has to return to its former state. Therefore, to describe proactive updates a knowledge state must be extended with a memory.

This can be done by giving the agent complete recall [24]. We will do this as follows: The agent remembers the initial state and keeps a stack of all updated formulas. Any previous state can
then be retrieved by removing the unwanted updates from the stack and updating the initial state with the formulas still on the stack.

**Definition 6.3**: A knowledge state with complete recall is a tuple \( \langle \sigma, M \rangle \), where \( \sigma \) is a knowledge state as defined before, and \( M \) a memory.

A memory \( M \) is a tuple \( \langle \sigma_i, L \rangle \) where \( \sigma_i \) is the initial knowledge state and \( L \) a list of sentences.

Two operations, adding and deleting are defined on the memory. Adding involves inserting a sentence to the end of the list \( L \). As we will use memory to trace the updates made, we have to record both proactively and reactively updated sentences.

To return to a previous state \( \sigma_p \), we look for the sentence \( \phi \) in the memory that was proactively updated to \( \sigma_p \). We then remove \( \phi \) from the memory list and all sentences that come afterwards. The initial state, kept in memory is then updated with all sentences in the new memory list. In addition, it may be necessary to update the state with the negation of \( \phi \) in order to prevent it from being chosen again.

We will not elaborate further on the formal aspects of the use of memory in the representation of knowledge states. here, we would like to draw attention to the fact that a memory is needed for proactive behaviour and that its use can be described in a straightforward manner.

### 6.5.5 Criteria and requirements

Knowledge states change when updates are made. State change ends when a solution has been found. We represent this as is follows: A state can be considered to be a final state when it accepts a given epistemic test. A knowledge state is final when all the candidate solutions in its associated solution state meet the criterion of the problem. A criterion describes when a candidate solution is regarded a solution to the problem.

For example, in the Master Mind game the criterion is given by three closed dots, indicating that all pegs have the right colours and are on the right position. Acceptance of this criterion in a solution state means that all solutions of the state are solutions of the problem. When more than one solution is required, quantification over the criterion becomes necessary.

Proactive updates are choices which are made in an attempt to reach a state which fulfills the criterion. Hence, the agent's proactive behaviour is goal-oriented. The criterion can be seen as the specification of a goal state, very much like a postcondition in the specification of programs.

We also use the notion of requirements to specify certain aspects of a solution. A requirement can thus be seen as a partial specification of the criterion. They can be useful in the representation of problems where the solution has to be constructed.

For example, in assignment problems one is looking for an assignment which satisfies certain constraints and is maximal with respect to a number of given preferences. Each constraint and preference can be seen as a different requirement, and the criterion is the expression which describes that an an assignment is a solution only if it satisfies all constraints, and is most preferred. Like the criterion, requirements can be represented as epistemic tests and the agent’s pro-active behaviour is aimed at reaching a state such that all these tests succeed.

In general, the framework we describe, stresses a distinction between knowledge about the solution and the specification of a solution. When solving the problem the two gradually move towards each other, and finally meet.
6.6 Describing problem solving

Problem solving can be described as a series of updates (both reactive and proactive) and epistemic tests, together with some flow of control. An agent starts problem solving with an initial solution state, containing all possible solutions and by successive state changes ends in a state in which the criterion of the problem is accepted. In the description of a problem solving method for the problem one can use the criterion to come up with a method. We will give some examples of how this works in the next section. A criterion is an epistemic test and may give hints how to reach a state which satisfies it.

6.6.1 Example: three classification criteria

In order to illustrate the notions explained above for a knowledge intensive task, we will describe three types of classification problems. It will be shown how an update system can be tailored to meet the requirements of each classification criterion. This will enable one to make systematic comparisons between them.

For each problem a simple method will be given. These methods will emerge quite naturally from the description of the criterion in an update system. Since updates and tests already provide some basic problem solving behaviour one often only needs to identify some iterations. For these proactive updates can be used. It will be shown that the nature of these proactive updates will differ among criteria.

Preliminaries

The goal of classification is to identify an object which is described by observations, as belonging to a certain class. The task ontology describes the vocabulary in which classification problems can be formulated. A detailed description of this vocabulary was given in chapter 4.

Next, we choose a simple propositional representation. Observations are atomic propositions of the form attribute=value. The knowledge which expresses that an attribute can have only one value is left implicit. The same representation for classification was used in chapter 3.

Class definitions are sentences in which a class implies conjunctions of attributes and value pairs. As an example consider the following class definition:

granite → grainsize=large ∧ origin=plutonic

Observations will be represented as attribute-value pairs, for example grainsize=small. The agent's domain knowledge consists of all class definitions. This knowledge will remain invariant during problem solving and is therefore a static part of the agents knowledge state. The dynamic part of the agent's knowledge state, its solution space, is built from the set of classes.

If C is the set of all classes then its powerset π(C) generates all possible sets of classes. As in update semantics these sets will serve as models. The dynamic knowledge forms the variant part of the knowledge state. The solution space will be used to keep track of changes over states.

The minimal knowledge state is given by the powerset of all atomic sentences of the language. This has to be updated with the domain theory, to arrive at the initial state. Let DT be the domain theory and W the minimal knowledge state. Then the state which captures the domain theory is given by W[DT].
The solution state of the minimal state $W$ is given by $\phi(C)$. If the domain theory $DT$ contains knowledge which excludes certain combinations of solutions then these are not part of the solution state associated with $W[DT]$.

As an example, consider a domain theory $DT$ with three very simple class definitions in the domain of igneous rocks.

- granite $\rightarrow$ grainsize=large $\land$ origin=plutonic
- basalt $\rightarrow$ grainsize=small $\land$ origin=volcanic
- diorite $\rightarrow$ grainsize=medium $\land$ origin=plutonic

These are part of the static part of the agent's knowledge state. The state also contains additional (static) knowledge expressing that identical attributes with different values are contradictory.

The dynamic knowledge is formed by possible solutions, and is represented as a solution state. The type of solutions is a class. In this example there are only three: granite, basalt and diorite which we abbreviate as $g$, $b$, and $d$. The minimal solution state, associated with the minimal state $W$, is simply the powerset of all classes:

$$\phi(g, b, d) = \{\{g, b, d\}, \{g, b\}, \{g, d\}, \{b, d\}, \{g\}, \{b\}, \{d\}, \emptyset\}.$$ 

The world $\{g, b, d\}$ expresses the possibility that granite, basalt and diorite are solutions simultaneously. In this example the domain theory, together with the constraint that identical attributes may have no different values, excludes some of these worlds.

The solution state with respect to the the initial knowledge state $\sigma = W[DT]$ then becomes:

$$\sigma_s = \{\{g\}, \{b\}, \{d\}, \emptyset\}.$$ 

In words this solution state tells us that either granite, basalt or diorite may form a solution, or that there is none. Observations are represented like the av-pairs which occur in the class definitions. Every observation enters the knowledge state as an update. Consider the observation origin=plutonic. As we update the solution state $\sigma_s$ with this observation, we get $\sigma_s[origin=plutonic] = \{\{g\}, \{d\}, \emptyset\}$.

The update has eliminated one possible world, being $\{basalt\}$. It is eliminated because origin=volcanic, occurring in the class definition, is contradicting the observation.

**Weak classification**

The criterion of weak classification can be stated as: a class is a solution if and only if it is consistent with the observations.

A class is consistent with an attribute when either one of the following two options hold:

1. The attribute as given by the observation does not have a value in the class definition.
2. The attribute value pair matches that of the class definition.

In order to deal with the first option an 'undefined' value has to be given to attributes which do not have a value in classes. The reason for introducing such a value has to do with the maximal consistency of the worlds that form a knowledge state: absence of an atomic proposition in a world is interpreted as its negation.

There are several ways to deal with undefined attributes. One is to incorporate it into the domain knowledge and state that its occurrence implies the disjunction of all other values of the attribute.
Another way is to check for undefined attributes in the method itself. When we update an observation to a knowledge state we lose the classes which are undefined for the attribute in the observation. In order to prevent this we update with the disjunction of the observation and its undefined variant. Notice that this mirrors the two options listed above.

Let \( a=v \) be an observation, \( c \) a class and \( \sigma \) the knowledge state containing the class definitions. Below we state three equivalent formulations of the criterion for weak classification:

\[
\begin{align*}
\sigma[a=v \lor a=\text{undefined}][c] \neq \emptyset & \iff \sigma \not\models (c \land (a=v \lor a=\text{undefined})) \iff (\text{1}) \\
\sigma \not\models (c \land (a=v \lor a=\text{undefined})) & \iff \sigma[a=v \lor a=\text{undefined}] \not\models \Diamond c \iff (\text{2}) \\
\sigma[a=v \lor a=\text{undefined}] \not\models \Diamond c & \iff (\text{3})
\end{align*}
\]

The first formulation (1) states that a class \( c \) is a solution if we update \( \sigma \) with the observation (in disjunction with the undefined value) followed by an update of \( c \) then the result should not lead to the absurd state. The second formulation (2) puts this in the form of an epistemic test \( \Diamond \). The test is accepted (succeeds) when the conjunction of \( c \) and the observation passes the test with respect to \( \sigma \). The final equivalent way of putting this (3) chooses to update the observation and test the class \( c \) for consistency.

Remember that solution states contain possible worlds of classes. A class (an atomic proposition) is consistent with respect to a state iff it is in at least one world. Hence, after having updated the observations the union of all worlds of the resulting state contains all classes which are consistent.

Formulation (1) suggest to update the observation (with the disjunction of an undefined alternative) followed by a proactive update of a class. When this last update succeeds the class selected in the proactive update is a solution. The second formulation (2) does not mention updates. It defines the criterion entirely as a test for consistency. The third (3) suggests to update the observation and test a particular class.

Observations are seen here as providing information to the agent about the outside world and therefore they lead to reactive updates. Hence, when choosing one of the three formulations we prefer formulations (1) and (3) rather than (2), because in (2) the observations are not updated. In addition, we let the agent select a class which it will test for being a solution proactively. Formulation (3) expresses this most clearly.

The three formulations of the criterion are logically equivalent but can also be interpreted as having a distinct procedural meaning. One can look at such formulations as a high-level, initial description of a method. Viewing formulation (3) as a high-level specification of a method, it turns out to be a variant of the select-and-test method, presented earlier in the Master Mind example. Here the test part takes the shape of a test for consistency of the selected class.

However, note that no matter which class is selected proactively, all will pass the test. Since every proactive update is consistent the test becomes redundant. Observations will reduce the solution state, removing inconsistent classes with every update. Hence, weak classification can be seen as a purely informational or reactive method, where no decisions have to be made by the agent. When only one solution must be produced the agent may choose one proactively. Still, the criterion of weak classification allows any choice from the solutions found so far.

Note, that in the depiction of the method in figure 6.5 a few assumptions are made. First, it is assumed that updating the observations never leads to the absurd state - i.e. a contradiction. This
Figure 6.5
A schematic representation of weak classification, producing one solution.

is a fair assumption and should be part of a classification task ontology. A second assumption is that after updating the observations a state is reached with at least one candidate solution. If this assumption is dropped the method should halt whenever this occurs.

6.6.2 Weak classification with preferred solutions

The criterion of weak classification defines a solution to a classification problem as a class which is consistent with the observations. The problem can be strengthened somewhat by adding knowledge of preferred solutions in combination with observations. This knowledge will be of the form: if obs is observed then class will be preferred.

We will show how this can be modelled in the update system just described. We consider again the example knowledge base of class definitions:

\[
\begin{align*}
\text{granite} \rightarrow & \quad \text{grainsize}=\text{large} \land \text{origin}=\text{plutonic} \\
\text{basalt} \rightarrow & \quad \text{grainsize}=\text{small} \land \text{origin}=\text{volcanic} \\
\text{diorite} \rightarrow & \quad \text{grainsize}=\text{medium} \land \text{origin}=\text{plutonic}
\end{align*}
\]

In addition we add knowledge about preferred solutions. We like to express that when some observations are made some classes are preferred to others. For example: if the origin is plutonic we prefer granite to diorite (and other plutonic rocks). This can be represented by using the \text{n} operator from section 6.4.3. The resulting expression then becomes: \text{n(}\text{origin}=\text{plutonic} \rightarrow \text{granite} \text{)}

To see what the knowledge states look like for this type of classification, consider the knowledge state containing the class definitions and this one sentence.

\text{origin}=\text{plutonic} \rightarrow \text{granite} \text{ is equivalent to } \neg \text{origin}=\text{plutonic} \lor \text{granite} \text{. Hence all worlds containing granite or lacking origin}=\text{plutonic} \text{ are optimal in the new state. When no observations are made we want all classes to be equally preferred. When origin}=\text{plutonic} \text{ is observed and updated to the state we want granite to be the most preferred class.}

Logically, as a result of the update with origin=plutonic, all optimal worlds of the new state will contain granite. Notice also that when origin=volcanic is observed (and updated) instead, then by using the class definitions all worlds containing granite are removed from the knowledge state, just as in weak classification. This is because the same attributes with different values are defined as being inconsistent.

The most preferred class in a given state \(\sigma\) can be described as the one which occurs in all optimal worlds of \(\sigma\). This can be tested by making use of the operator \(p\). Remember that the test \(p(\phi)\) succeeds with respect to a state \(\sigma\) when \(\phi\) occurs in all optimal worlds in \(\sigma\). If after updating
Fig. 6.6
A method for weak classification with preferential knowledge.

the knowledge state with the observations $p(c)$ is accepted for some class $c$ then this class is both consistent and most preferred.

The criterion for classification with preferred solutions can then be put as follows:

$$\sigma[a=v \lor a=\text{undefined}] \models p(c)$$

or equivalently:

$$\sigma[a=v \lor a=\text{undefined}][p(c)] = \sigma$$

From this criterion we can again formulate a method. A method description is given in figure 6.6. It starts off similar to the one for weak classification, updating observations as they are made. However, notice that here the worlds in each knowledge state are ordered by the preferential knowledge and that we are working in a different update system. Updating observations removes some solutions, exactly like in weak classification. Next, a proactive update is made to a state containing all optimal worlds. This reflects the part in the criterion which mentions the $p$ operator.

This update is labelled proactive in figure 6.6. Selecting all optimal worlds is not an indeterministic operation in the logical sense but what counts as an optimal world is a non-monotonic in the following sense: if $w_1 \ldots w_n$ are optimal worlds of $\sigma$ then they are not necessarily optimal in a state $\sigma' = \sigma[\phi]$ where $\sigma[\phi]$ is an informative update.

For example, consider an agent making all observations first and then proactively selecting all optimal worlds of the current state. For example, in the case of medical diagnosis a physician may select a number of most preferred diagnoses on the basis of a number of observations. When new information comes in after this selection has been made, the agent has to backtrack to the previous state, update the new information and select the optimal worlds again. This possibility of backtracking is not shown in the figure. It is assumed here that all observations have been made.

A class is a solution if it occurs in all worlds of this new state. The next move is then to choose a single class from a world and use it as a proactive update. The new state contains a single
solution class and satisfies the criterion.

6.6.3 Classification with inconsistent observations

A third classification criterion which will be described here, deals with handling inconsistent observations with respect to class definitions. In classification problems it frequently occurs that no class is consistent with all observations. In such situations it could be desirable to return the class which has the least number of inconsistent attributes relative to the observations made.

Several variations on such criteria exist. These are described in chapter 4. For example, one could demand that some ax-pairs should always be consistent whereas others are allowed to be inconsistent with respect to the observations. We will not go into the details of such alternative criteria here. However, we would like to make clear that they can be described in an update system very similar to the one explained above.

The first point to note is that the criterion which describes the class with the least inconsistent attributes as the solution, defines another preference structure. One can partially order all classes in such a way that classes with more inconsistent attributes are more minimal than those which have less. Finding a solution means to return the minimal element in this preferential ordering.

The second point is that this preference structure is dynamic and changes as more observations are updated to the state. This is different from the preferential structure in the previous classification criterion we discussed. There the preference structure was given by preferences in the domain knowledge and remained static during problem solving.

A crude method for finding a class which has the least inconsistent attributes can be described as follows:

If $\text{Obs} = \text{obs}_1 \ldots \text{obs}_n$ are the observations then first a class $c$ is looked for such that $\{c\} \cup \{\text{obs}_1 \ldots \text{obs}_n\}$ is consistent. If there is one, $c$ is a solution. If not it is checked whether there is a $c$ for which $\{c\} \cup \text{Obs} \sim \{\text{obs}_i\}$ (for some observation $\text{obs} \, 0 < i \leq n$) is consistent. If still a solution has not been found $\{c\} \cup \text{Obs} \sim \{\text{obs}_i, \text{obs}_j\}$ $(0 < i, j \leq n \text{ and } i \neq j)$ etc.

As we have seen above, in update semantics one can describe a system in which the worlds are ordered. The update system with the operators $\diamond \text{p, n}$ and $\text{n}$ can be used to describe this method.

To do this we represent the class definitions simply as implications. When an observation is made it is updated to the knowledge state. However when $\text{obs}$ is an observation and $\sigma$ a state then we update $\text{obs}$ to $\sigma$ as $\sigma[n(\text{obs})]$. The $\text{n}$-operator results in an ordering of worlds within the state such that those worlds in which $\text{obs}$ occurs are preferred to those which do not contain $\text{obs}$. A class is consistent with all observations made in this manner if it is consistent in the state formed by all optimal worlds (which are also normal worlds at this stage).

If there is no such class, the state should be updated in such a way that all optimal worlds are removed from the state. Again it is checked whether a class is consistent in the optimal worlds. If not the process repeats itself.

As an example consider again the definitions of igneous rocks:

$\text{granite} \rightarrow \text{grainsize}=\text{large} \land \text{origin=}\text{plutonic}$
$\text{basalt} \rightarrow \text{grainsize}=\text{small} \land \text{origin=}\text{volcanic}$
$\text{diorite} \rightarrow \text{grainsize}=\text{medium} \land \text{origin=}\text{plutonic}$

If $\text{n}(\text{origin=}\text{plutonic})$ is updated then a preference for worlds containing $\text{origin=}\text{plutonic}$ is realized. This does not mean that for example, granite is true in all optimal worlds. However it
does mean that are some optimal worlds in which granite occurs. The same is true for diorite but not for basalt.

To test whether a class occurs in an optimal worlds one can make use of the operator $\diamond p(c)$, defined in section 6.4.3. Now, suppose the observation $\text{grainsize=tiny}$ is made. This is updated as $\text{n(grainsize=tiny)}$. At this stage none of the three rocks occur in any of the optimal worlds.

For example, granite implies $\text{grainsize=large}$ which is contradictory to $\text{grainsize=tiny}$ which is true in all optimal worlds. However, both granite and diorite are still preferable to basalt since they are consistent in worlds higher up in the ordering.

The criterion for this problem variation can be expressed by using the $p$ operator which is used to test whether classes occur in some optimal worlds. Or equivalently:

\[
\sigma[a=v \lor a=\text{undefined}] \models p(c)
\]

Note, that this is criterion is similar to the previous classification criterion: classification with preferred solutions. Where we used $p$ previously we now use the weaker $\diamond p$. In words: instead of checking whether a class occurs in all optimal worlds we now look for a class which occurs in some optimal worlds.

![Figure 6.7](image.png)

**Figure 6.7**

Classification for classes with least consistent attributes.

The method is depicted in figure 6.7. The knowledge state gets ordered by observations which are all prefixed with the $n$ operator. The result is an ordered state. Then it is checked whether there are classes such that $\diamond p(c)$ holds. This is shown as a test with as input the criterion. If this test succeeds then state is proactively updated such that the solution is produced.

If no class is present in an optimal world then the test fails and backtracking takes place. Optimal worlds are now removed and the same procedure is repeated for the next optimal worlds etc. until a solution has been found.
6.7 Classification criteria compared

The last three simple examples are meant to show that problem solving behaviour follows quite naturally from the specification of the task in terms of the vocabulary of the framework. It also allows for a systematic comparison between these tasks, as we will now show.

Weak classification can be described in an intuitive manner by a propositional update system with an epistemic test for consistency. The static domain knowledge can be represented as is common in classification systems. The representation of a knowledge state consists of worlds without preferential structure. The criterion of weak classification is formed by the epistemic test for consistency, and updates take the role of observations.

Solving a problem of weak classification can be described without any reference to proactive behaviour of the agent. All that is needed is an update of observations to the agent’s knowledge state. This alone will prune inconsistent classes from the solution state. Stating that weak classification can be described purely reactive also means that from a knowledge level perspective the task is purely deterministic.

Extending weak classification with knowledge about preferential solutions changes the representation of the static domain knowledge. The logically machinery changes as well and we move to another update system. We have shown how the $p$ and $n$ operators were used in the representation of preferential domain knowledge.

This preferential structure remains static during problem solving. Observations are represented as propositional updates, exactly like in weak classification. Their role is again the pruning of classes from the solution states. The criterion is now formed by the epistemic test for acceptance in optimal worlds, $p$.

This system is clearly different from the one used for the normal, weak variant. Its main characterizing feature is the use of a preference ordering. This leads to changes in domain knowledge and criterion.

The third criterion for classification: preference for classes with least inconsistent attributes, also makes use of a preference structure. Hence, the knowledge state is ordered here as well. But no adaptations to the static domain knowledge are needed here, as compared to weak classification. The updates of observations take a different role and are no longer propositional. The criterion is described by the epistemic test $\Diamond p$.

The two last problem variations differ in the representation of the knowledge state. In "classification with preferred solutions" the preference structure is static and part of the domain knowledge. In "classification with inconsistent observations", the preference structure is given by means of observations and changes dynamically.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Static knowledge</th>
<th>Knowledge state</th>
<th>Updates</th>
<th>Criterion</th>
</tr>
</thead>
<tbody>
<tr>
<td>WC</td>
<td>Class defs.</td>
<td>Propositional</td>
<td>Obs - atoms</td>
<td>$\Diamond (c)$</td>
</tr>
<tr>
<td>WCP</td>
<td>Class defs. + n-prefs.</td>
<td>Statically ordered</td>
<td>Obs - atoms</td>
<td>$p(c)$</td>
</tr>
<tr>
<td>WCI</td>
<td>Class defs</td>
<td>Dynamically ordered</td>
<td>Obs - n(0)</td>
<td>$\Diamond p(c)$</td>
</tr>
</tbody>
</table>

Table 6.2
A comparison of the example classification problem descriptions.

Table 6.2 summarizes a comparison between the three classification variations. The description of these examples allows one to compare different task configurations in a systematic way.
Each may require a different representation of domain knowledge, observations or criterion. And each may require a particular logic to express it.

In table 6.2 weak classification is abbreviated as WC. The static part of the knowledge states is formed by the definition of the classes. The formal representation of the worlds of a knowledge state is by means of sets. There is no relation ordering these worlds and the underlying update system is propositional. The updates are observations and are atomic propositions. Finally the criterion whether a class \( c \) is a solution is given by the epistemic test \( \Diamond(c) \) for consistency.

In this way one can compare problem variations in a systematic way. Problem descriptions vary in the structure of the domain knowledge, the nature of the updates and knowledge states and of course the criterion. By modifying one of these "parameters" one can describe other problem variations quite easily. For example, one can provide "strong" versions of the above criteria as follows. For weak classification one has to replace the test \( \Diamond c \) with \( \square c \), which means that \( c \) should be an explanation for the observations. To see this remember that acceptance of \( \Diamond c \) means that \( c \) is present in all worlds, whereas acceptance of \( \Diamond c \) means that \( c \) occurs in some worlds. This difference can be used to express weak (consistent with the observation) and strong (implying the observations) classification.

In the third criteria a similar change is needed, replace \( \Diamond p(c) \) with \( p(c) \). The second criterion involves a change in the static domain knowledge. In fact, one has to make sure that the preferential knowledge of a class \( c \) is expressed as:

\[
\mathbf{a}_i = v_i \land \ldots \land \mathbf{a}_l = v_i \rightarrow c
\]

provided that the class definition of \( c \) is \( c \rightarrow \mathbf{a}_1 = v_1 \land \ldots \land \mathbf{a}_l = v_l \). This "completion" is necessary to make sure that \( c \) will imply all observations when selected as a solution. This is required for "strong classification" criteria, as explained in chapter 4.

### 6.8 Discussion

The framework presented here enables one to describe tasks and problem solving in terms of what an agent knows about problem solving. We view knowledge as a semantic notion and have chosen a semantic framework for its representation. For a description of the logical machinery we have made heavy use of Veltman's update semantics and adapted it in order to be able to account for problem solving behaviour of a knowledge level agent. We introduced such knowledge level constructs as proactive updates and memory.

The semantic point of view allows us to abstract to knowledge level descriptions. To summarize the approach we have taken, consider the following template of actions. First, when given a problem we identify the type of a solution. For classification this is a class. (Its definition is part of the task ontology.) Next we identify the static domain knowledge, the updates and the criterion of the problem.

We have shown how the specification of classification problems can lead to a description of problem solving behaviour in terms of changing knowledge states. The nature of the static domain knowledge, the formal structure of the knowledge states, the nature of updates and the criterion can be seen as key properties in the description of a problem. With a slight abuse of vocabulary we refer to such a description as "the semantics of the problem".

Different requirements on a solution will lead to different tests, updates and different structured knowledge states. We have illustrated this by looking at three classification criteria. Other classification criteria (see chapter 4) can be described in a similar way.
The framework described takes a model-based approach. In practice this means that we begin by making a knowledge-level analysis of the problem, identifying the static and dynamic knowledge an agent has access to. This results in an idea of what the structure of the knowledge states should look like. One can then choose or define update operations and describe problem solving in an update system. Formally, an update system can be seen as the semantics of some logical language. We have seen three illustrations of this in section 6.4.3.

Hence, where knowledge representation usually begins by selecting a proper representation language, we start with the identification of the structure and end up with a semantic description of a representation language.

The framework presented here has focussed on the knowledge-level description and functionality of methods. Means of making a representation of knowledge states computationally attractive are topics for further study. As we have indicated, characteristic models and the use of tableaux are two promising ways of pursuing this.