Systems that detect and repair their own mistakes

Khemdut Purang

August 21, 2001

Contents

Abstract

Making mistakes is an inescapable aspect of everyday life. We constantly make mistakes, recognize them and try to correct them. Mistakes are inevitable because of the incompleteness of our knowledge of the world, its inherent uncertainty and its being in a constant state of change. We can never know for sure that what we know is true and the actions that we take based on these beliefs can therefore be misguided. Sooner or later we act based on some false belief or the world changes in an unexpected way and we fail to achieve our goal. But the fact that we can recognize and repair these errors mitigates their effects. Software systems face the same problems. The difference is that they do not ususally have as robust a capability as we have to detect and respond to their mistakes. This is part of what makes them seem brittle and user-unfriendly. This problem is not likely to get any better as the systems exhibit more complex behaviors in more realistic domains.

Our work begins to address that problem by focusing on the computational capabilities required of software systems for them to be able to autonomously recognize and respond to their own mistakes. We study in particular, mistaken beliefs, intentions and actions in agents that have some goals to achieve. Intuitively enough, the basic capabilities required are an ability to inspect their past behavior and computations and the past states of the world and to use that to determine their future behavior. These abilities are not typically available in software systems.

We have implemented a general logical framework in which one can specify the behavior of an agent that supports this kind of representation and computation . We have implemented agents that detect and respond appropriately to their mistakes in some aspects of language processing. We have also implemented a system that handles its mistaken beliefs in any domain that can be described using the language of non-monotonic logic. This system was tested on a test suite that we compiled from examples of non-monotonic reasoning in the literature. We finally provide a design of the representations and algorithms for handling mistakes in an agent that acts in the world and has mistaken beliefs, intentions and actions. Implementing such an agent is the next step in this work.

Part I

Mistakes and active logic

Chapter 1

Mistakes

> Pooh always liked a little something at eleven o'clock in the morning, and he was very glad to see Rabbit getting out the plates and mugs; and when Rabbit said, 'Honey or condensed milk with your bread?' he was so excited that he said, 'Both' and then, so as not to seem greedy, he added, 'But don't bother about the bread, please.'

As soon as he said "Both", Pooh [127] realized that saying that made him seem greedy and that was a mistake. So he immediately tried to correct the mistake as best he could with comical effect.

Mistakes are made by highly trained people in more dramatic circumstances too as this newspaper excerpt [147] shows:

> A National tower controller cleared a Piper PA-27 to land on Runway 4, then mistakenly gave a US Airways Boeing 737 clearance to take off on Runway 1, which intersects with Runway 4, less than 1,000 feet from the Piper's touchdown point.
>
> Nineteen seconds later, the controller caught the error and realized the two planes could meet at the intersection. The controller ordered the Piper pilot to abort his landing.
>
> But the pilot, Ronald W. Zborowski of Eighty Four, Pa., said in an interview that the controller was using the wrong call sign for his plane and he did not realize at first that the controller was talking to him.
>
> Instead, Zborowski said, he saw the US Airways plane beginning his takeoff roll nearby to his right, and "stood on the brakes" to come to a stop. "Had I proceeded with a normal landing, there was a danger of a collision," he said.
>
> The Piper stopped 200 feet short of the intersection as the jet roared past.

If it had been a software system that had put the planes on a collision course, would it have realized its mistake? And would it have attempted to correct it? What would it take for software systems to have that ability?

AI systems are notoriously fragile, breaking completely once they get outside their design specifications. Two approaches to this problem are (1) to design and build the systems so that they accurately and completely model all the circumstances they can find themselves in; (2) to accept that there is always the possibility of failures and to recognize and correct them when they occur. A mixture of the two seems to be required for more flexible AI: we need to more precisely model the world, but we also need to handle failures of the system when they occur. In this work we investigate the second approach–how to design resource-limited agents that operate in non-trivial worlds so that they can themselves detect and respond to the inevitable mistakes that they make.

We do not propose to design and build a system that would act like the controller mentioned above. This would be a long range undertaking involving many aspects of AI. Instead, we focus on the design of the mechanisms for error detection and correction that such a system would need. We illustrate that this is a viable approach with a few implemented systems that have been tested on simpler examples of mistake than the above near-collision.

There has not been, to our knowledge, any previous work that specifically addresses the issues involved in detecting and handling mistakes across a large range of domains in the unified way we propose. Work in conditional planning and plan execution with failure address similar issues as far as agents devising and executing plans are concerned. Non-monotonic logics and belief revision systems address issues related to those of agents making mistakes in their beliefs. Mistakes in dialog and language have been addressed in some work on miscommunication. Each of these approaches involves specific techniques that are effective in their domains and for their purposes. However, none of them presents a unified and practical view of agents detecting and handling their mistakes as is our goal.

This unified approach is based on active logic [48]. We designed, implemented and tested Alma/Carne, a general implementation of active logic (See chapter 2). Previous work on active logic required a new logic to be built to solve the specific problem of concern. Alma/Carne is a basis on which active logic applications can be built. This facilitates the development of new applications and allows for reuse of parts of solutions across domains. Alma/Carne is time-situated and provides an extensive set of meta-logical predicates and operators. It has been tested on a large number of test problems during development and is being used in some projects. Several formalisms, for example[164, 122, 44, 39, 13, 90, 6, 15, 11, 99, 166], have one or more of the distinctive properties of Alma/Carne but not all of them. As will be seen in the rest of this work, these capabilities are essential for our goals.

Conditional planning, architectures for plan execution with failures and related systems [139, 186, 68, 62, 81, 80, 85, 7, 192, 192, 191] consider the problem of an agent planning and executing plans in circumstances where there may be mistakes. Some shortcomings that we see, in general, with these systems are: (1) all possible mistakes have to be predefined in the plan; (2) the methods of detecting these mistakes are fixed and have to be defined in the plan; (3) this results in large plans that are cumbersome to derive and execute. Further, these systems do not explicitly consider mistaken beliefs and the relations between mistaken beliefs, intentions and action and therefore miss opportunities to detect and also prevent mistakes by the agent. Our approach presented in chapter 10, addresses these issues.

Non-monotonic reasoning and belief revision systems [110, 108, 157, 130, 116, 173, 43, 162, 40, 41, 149, 170, 25, 65] address problems related to mistaken beliefs. We see the following problems with non-monotonic reasoning formalisms in general: (1) they do not allow for change in the knowledge base; (2) they are not time-situated; (3) they are not computable; (4) they do not allow meta-reasoning; (5) they do not tolerate contradictions. Belief revision systems allow dynamic change of their databases unlike non-monotonic systems, but still have the other problems. Our approach to mistaken beliefs uses the language of non-monotonic logics but addresses these problems. (See chapters 7 and 8). This has been tested on a large number of problems collected from the literature (see chapter 9). These are problems put forward to illustrate non-monotonic logics or to contrast logics according to the problems one and not the other could solve. Of 96 problems collected, 46 were solved by our system, 23 were similar to the 46 and could be solved but were not attempted and 24 (25%) could not be solved.

Work in miscommunications in natural language dialog [120, 82, 185] is another area where mistakes are considered in the literature. We applied our approach to two specific cases of mistakes in language processing: presupposition projection [78] (see chapter 4) and implicature cancellation [74] (see chapter 5). In the implicature cancellation case, our approach better models the evolution of the knowledge base with time than alternative approaches do. In the presupposition projection case, our modification of a widely accepted algorithm [78] to reason about mistakes in a class of dialogs resulted in more intuitive performance of that algorithm.

A point that bears emphasis is that our approach to mistakes and the logic that we implemented for that provide a uniform basis on which one can build specific instances of mistake detection and handling in different domains. This has been demonstrated in each of the domains mentioned above and is detailed in the rest of this document. This provides evidence that the methods to detect and handle mistakes we develop here are domain-independent.

1.1 Scope of this work

Note that we focus on agents detecting, on their own, that they have made a mistake, and repairing these. There are related perspectives that we do not adopt: 1. that of the designer of the agent determining whether or not the agent performs according to the designer's intentions; 2. that of a user of the agent who determines that the agent does not satisfy the user's needs. These perspectives are all related–ideally each perspective will generate identical mistake sets. Then, the desired response of the user, the designed response and the actual response of the system coincide for all situations. However, the designer of the system might not know what a user of the system will expect and the system itself may be built so that it cannot detect and respond to some of the situations it finds itself in.

We take a rather wide view of what counts as an agent by including any artifact that can be said to have beliefs and intentions and is capable of acting in some way, even if it is only to modify its internal structure. We also take a liberal view of what counts as an agent having beliefs, goals, and intentions by adopting the intentional stance [42, 111]. As long as we can coherently attribute beliefs and intentions to an agent, we take the agent as having these mental states. There also have to be internal states or processes that we can identify to which these mental states could be attributed.

This helps us have a general account of mistakes for a wide range of systems. The complexity of the design of the agent as well as the environment the agent operates in varies widely in complexity. So do the kinds of mistake that the agent can commit and the appropriate response to these mistakes.

The following examples can help clarify the scope of our investigations:

- *A vending machine rejecting a genuine dollar bill*. This is surely a mistake of the machine from the point of view of the user, and perhaps from the point of view of the designer, but it does not seem to be a mistake from the point of view of the machine because it does not seem to recognize that it has made a mistake in rejecting a genuine bill and does nothing to recover from that mistake. This is not the sort of behavior we are concerned in

- *A CD player that mutes on playing back a CD*. A CD player has to read data off the disk. However, things can go wrong: the data may not be readable, or it may fail integrity tests. On failure, CD players typically try to interpolate the missing samples, or mute the output for these samples. From the intentional stance, the CD player expects good data to be coming off the disk, but when that assumption is falsified it realizes there is a problem. Its first response is to try to recreate the data by interpolation. In this case the interpolation fails and the player then resorts to muting the output for the missing samples. On realizing that there is a mistake, whether on its part or because the CD being played is out of specifications, the player tries to repair the error before it gets to the output. The repair is not always perfect but this response seems more adapted to failures than the vending machine's.

- *A computer game monster that runs away from the player but when cornered, attacks.* Here the belief one could attribute to the monster is that it has an escape route, and given that, it runs away. If the path is blocked, it seems to recognize that and changes its behavior. In this case, it seems clearer that the monster recognizes that something unexpected has happened and responds to that event by changing its behavior. This could be seen as recognition by the monster that it had made a mistake and a response to the consequence of that mistake–that it has put itself in great danger. This is different from the previous case where the basic behavior of the CD player did not change in interesting ways. The monster seems more responsive to its failures than the CD player.

- In the near crash above, something like the following presumaby happened. The controller intially gave landing and take-off instructions to the two planes believing there was no conflict. He then found that a possible consequence of his (or her) instructions was a collision. This involved reasoning about his past action and their effects on the world. He recognized a collision as being undesirable and searched for a way to avoid that. He could not undo his past actions but tried to do that by asking the Piper not to land. It seems though that he did not know that it was already on the ground and this response was mistaken. He was doubly mistaken when he used the wrong call sign for the Piper.

These examples illustrate that there is a variety of agents to which we can at least attribute some form of mistake, and there is a wide range of responses we would judge appropriate, although we would not always associate agenthood or mistakes to these artifacts. It does seem appropriate for the CD player to interpolate data and mute the ouput when data is missing rather than rejecting it on the first error. We would think the monster to be particularly stupid if it tried to run through the wall (if it did not have special wall-penetration capabilities), but attacking when cornered seems to be a good response. The traffic controller's response to the original response was mistaken: he should have verified the position of the Piper and he should have known the call sign. This was perhaps the result of working under the threat of an imminent disaster. This response, though flawed, seems like the right kind of thing we would expect.

Another aspect of dealing with mistakes that we investigate then is the range of types of possible responses to mistakes. Not all agents need to have the greatest degree of flexibility to deal with mistakes and that apparent limitation can be very helpful given the limited resources of real agents.

As agents and software systems in general get more complex, the range of errors they could face will increase even faster. It will not be practical for the system to have precomputed and hardwired responses to the mistakes as with the simpler systems we have seen above. The CD player does not have many behaviors and does not have to respond to many unexpected events, so it can use a simple chip-based controller (for example the SAA7335 [146]). An increase in the complexity of the system requires a different kind of reaction to mistakes rather than more of the same kind of action though. The response has to be flexible and handle any situation the agent finds itself in, or at least a wide range of such situations. Computer game monsters have a wider range of behaviors and find themselves in more kinds of situations than CD players and have to respond to these appropriately if the game is to be successful. A Half-Life (a popular series of computer games) monster typically has a range of plans available and it executes just one at any one time. It has one current behavior and one response plan to deal with any failure in its current behavior [34]. This gives the monster more flexibility that the CD player but this is not sufficient for more complex agents in more complex situations. When the planes are about to collide, we want to have a variety of responses available where the choice of the response can involve complex but fast computations. The flexibility can be achieved if the system detects, identifies and reasons about the mistakes it makes.

Having agents diagnose and repair their own mistakes is especially important because as systems get more complex, it is harder for their designers to understand them and predict their mistakes in their interaction with the more complex environments that these systems will operate in (it's hard enough to predict the behavior in normal circumstances). It is therefore not going to be possible to have all the responses to mistakes precomputed and ready to be used. In fact, the recognition and classification of the mistake itself is likely to be much more complex so that even if we had a response for every mistake, it would be hard to respond appropriately. The solution we propose is to give the system the basic tools to deal with mistakes and let it devise its response to mistakes as they occur.

In the rest of this chapter, we first discuss the inevitability of mistakes and the importance of handling mistakes. We then define mistakes and describe some of their properties. Our discussion is meant to result in a practical set of tools for generating the behaviors described above and should not be seen as a philosophical analysis of mistakes. We next consider how an agent could detect and respond to mistakes and the representations needed for that. We then sketch out the rest of the dissertation.

1.2 Mistakes are inevitable and have to be handled

Mistakes are pervasive. We can almost say that if a system cannot make mistakes, it is not doing anything very interesting. Several factors contribute to making mistakes inevitable in an agent that operates in a non-toy domain:

- **Incompleteness** It is unlikely that one can represent all of the facts and relations relevant to the behavior of an agent in its KB. Not doing so can cause errors: if we don't know that there is a tiger behind the door, we might open it which would be a grave mistake.

- **Uncertainty** Even if that information that is represented in the agent, there is likely to be a degree of uncertainty associated with it. It is not the case that *all* birds fly, but *typically* birds fly. If all the agent knows is that Joe is a bird, and that birds typically fly, the best conclusion it can come to is that Joe flies. But Joe might be one of the exceptions and not fly. Believing that Joe flies is then a mistake.

- **Change** As time goes by, events occur that are controlled by other agents and the information that the agent has which that was true in the past can become false without the agent being aware of it. Even if the change is a result of the agent's actions it may still be unaware of it because of incompleteness in its knowledge or because of its resource limitations. The traffic lights may be green, and the agent correctly believes it to be green. If the lights then change to red but the agent does not notice it, that belief becomes mistaken and the agent may cross the street.

- **Resource limitations** The agent is likely to have a large amount of information and a limited processing capability. A large amount of information can be computed from the information that the agent already has. A real agent will consume resources (including time) in computing that new information. There may not be sufficient resources to compute some relevant piece of information in the time available. The rules of chess are finite and well-defined so that there are no problems of incompleteness or uncertainty. The games are of finite length and the changes to the board are known to the agent, but an agent will not in general be able compute the best move to make for lack of computational resources.

- **Time** Related to the above is the time that the agent has available to respond to changed situations. It might have to take actions before it has had the time to process all relevant information and that can lead to mistakes. The agent may be able to compute a very good chess move, if not the optimal move, given enough time, but if it does not have time, it is possible for it to make a mistake.

Each of these factors can cause the agent to make a mistake. It does not seem likely that the possibility of mistakes can be eliminated unless we retreat to the simplest of toy domains, or the agent refuses to take any action or represent anything. Making mistakes seems to be an inherent aspect of the behavior of agents in interesting enough domains.

1.2.1 Handling mistakes

It is obvious that an agent should respond appropriately to errors, even if the response may occasionally be itself mistaken. If it does not do so, it is not likely to accomplish its goals and may cause further problems in the world.

Apart from the effectiveness of the agent, the absence of failure responses that are appropriate to the behavior of the system can lead to frustrating experiences for users of those systems. A vending machine refusing to accept the only dollar bills you have is not a pleasant experience.

A related factor is that the perceived intelligence of a system seems linked to its ability to deal with errors. The more flexibly a system adapts to new circumstances, including those where it made mistakes, the more intelligent it is perceived to be. Vending machines are not perceived to be particularly intelligent. Game monsters, however, do sometimes at least, succeed in outwitting us.

People make mistakes all the time, yet can generally function well enough not to be overwhelmed by them. People even use mistakes to their advantage: we recognize that we have made a mistake, we might repair it and we might learn from that mistake and try not to repeat it. We need not be aware of all the mistakes we make or never commit the same mistake twice, but we are not oblivious to mistakes either. The mistakes we make help us build a better model of the world and that allows us to better operate in it. Similarly, intelligent agents should have the capability to recognize and respond effectively to mistakes even though they might not always be able to do so.

1.3 What is a mistake

We start by making more precise what we mean by the term “mistake”. The sense of mistake that we are after is one that will be useful for an agent that needs to react to mistakes it encounters, rather than dissecting the meaning of the term. We ultimately want a notion of mistake that can be easily computed by the agent even though it may not always be accurate (it is then itself mistaken).

1.3.1 Dictionary definition

The definitions in Webster’s Third New International Unabridged Dictionary [189] for “mistake” are:

> mistake
>
> n 1 : a misunderstanding of the meaning or implication of something
>
> it is a mistake to think that the supreme or legislative power of a commonwealth can do what it will —John Locke it is a great mistake to think that the bare scientific idea is the required invention —A.N.Whitehead
>
> 2 : a wrong action or statement proceeding from faulty judgment, inadequate knowledge, or inattention : an unintentional error
>
> it would be a mistake , however, to drain all bogs —Boy Scout Handbook gave him a ten-dollar bill in mistake for a one
>
> 3 law : an erroneous belief : a state of mind not in accordance with the facts
>
> syn see ERROR
>
> — and no mistake : SURELY, UNDOUBTEDLY he’s the one I saw, and no mistake

“Error” in turn is defined as:

> error
>
> n -S
>
> [ME errour, fr. OF error, errour, fr. L error, fr. errare to err] 1 a : an act or condition of often ignorant or imprudent deviation from a code of behavior : violation of ritual holiness, moral rectitude, or social convention : SIN
>
> entice with licentious passions of the flesh men who have barely escaped from [.....] error —2 Pet 2:18 (RSV) : OFFENSE, FAULT the official’s errors of nepotism and acceptance of large gifts from lobbyists
>
> b : an act involving an unintentional deviation from truth or accuracy : a mistake in perception, reasoning, recollection, or expression
>
> made an error in adding up the bill gunnery errors
>
> c : an act that through ignorance, deficiency, or accident departs from or fails to achieve what should be done
>
> got lost when he made the error of turning left at the fork an error of judgment the error of writing last year’s date early in January
>
> ...
>
> 3 : something (as a misstatement or misprint) produced by mistake
>
> a typographical error

> specif : a postage stamp released for use that shows flaw in its manufacture (as in differing in color or paper from others of its issue and denomination)
>
> ...
>
> 6 : a deficiency or imperfection in structure or function : DEFECT
>
> an error in vision may cause headaches
>
> syn
>
> MISTAKE, BLUNDER, SLIP, LAPSE, FAUX PAS, BULL, HOWLER, BONER: ERROR indicates a deviation from correct, sanctioned, approved belief, procedure, practice, or course
>
> ... MISTAKE suggests a misunderstanding, wrong decision, or inadvertent wrong action; it may apply to the unimportant or momentary but does not always do so a mistake in reading the road map a mistake in admitting these students a mistake in copying the list
>
> ...

The main points of interest to us are that:

1. Mistakes or errors can involve beliefs, meanings, actions, perception, and reasoning. We are also interested in intentions as intentions mediate between the beliefs of the agent and its actions.
2. Mistakes are caused by inadequate knowledge, faulty judgment or by accident. We equate inadequate knowledge to incomplete knowledge; faulty judgment to making a decision before enough information has been taken into account; accident to unpredicted changes in the world.
3. Mistaken beliefs are beliefs not in accordance with the facts and mistaken actions are actions that fail to achieve their goal.

We can see generally that a mistake occurs when there is a mismatch between an object on the part of the agent and what it is meant to accomplish:

- A belief is mistaken when there is a mismatch between the belief and the corresponding fact in the world.
- An action is mistaken when there is a mismatch between the goal the action was intended to accomplish and the effects of the action.
- An intention is mistaken if there is a mismatch between the consequences of executing the plan intended and the goal of the agent.

1.3.2 A preliminary view

An aspect not explicit from the definitions is time. A belief is a mistake if *at the time* the belief was held, it was false. The same belief (we do not consider “eternal beliefs” [153]) can be mistaken at some times and not at others. If the agent believes that it is not raining, for instance, and a rain shower passes though, the fact that it is not raining changes truth value over time so that whether the belief that it is not raining is mistaken also varies. The same phenomenon holds for intentions and actions.

The picture that emerges is that a mental object or an action is mistaken at some time if there is a mismatch between what that object is meant to accomplish and whether that is accomplished at that time. This mismatch is caused by a deficiency in the knowledge or computations of the agent or by an unexpected event in the world.

We will therefore consider the following kinds of mistake:

1. **Mistaken beliefs** A belief the agent holds at time t_i is mistaken if the belief is not true at t_i.
2. **Mistaken intentions** An intention held by the agent at some time t_i to execute a plan at some time t_j is mistaken if the execution of the plan at t_j does (or will or did) not result in the goal desired.
3. **Mistaken action executions** An action or plan being executed at some time is mistaken if the execution at that time does not result in the expected outcome.

Note that if the agent represents that it has committed a mistake, then that belief itself is subject to being mistaken which allows the possibility of iterated mistakes. Also, the mistakes are centered on the agent. We consider only mistakes done by the agent doing the reasoning. However, these definitions require the agent to know what is true or whether execution of an action or plan will succeed. This requires an external omniscient view of the agent and of its domain to compute, and cannot be done by the agent itself.

1.3.3 A computable agent-centered definition

In keeping with our desire to have a notion of mistake that is from the agent's perspective and computable by the agent, we have to modify these definitions. The intuition is that the beliefs of the agent can change over time. This change in beliefs may imply that the old belief was mistaken. If the agent believed that Tweety could fly and later realizes that Tweety is a penguin and therefore cannot fly, its belief at that time, that Tweety could fly, was mistaken. A change in mind does not always imply a mistake, for instance, a change in the traffic lights may lead the agent to now believe that the light is green whereas it previously believed it to be red. This change in belief tracks a change in the world and so the initial belief that the light was red was not mistaken. The same applies to intentions and actions done by the agent.

So, for an agent, recognizing a mistake involves looking back in time and comparing its beliefs, intentions, and actions then to the situation it later believes to have been the case. We modify the above to have the following kinds of mistakes from the perspective of the agent. Note that we call these mistakes *recognized* to emphasize that they are to be computed by the agent making the mistake itself rather than from an external point of view.

1. **Recognized mistaken beliefs** An agent recognizes that a belief ϕ it held at time t_i was mistaken if at a later time t_j, it believes that ϕ was not the case at t_i.
2. **Recognized mistaken intentions** An agent recognizes that an intention it held at t_i to execute a plan at t_j is mistaken if, at some time t_k, $(t_k > t_i)$ the agent concludes that it is not the case that the execution of the plan at t_j will result or has resulted in the goal desired.
3. **Recognized mistaken action executions** An action or plan being executed at t_i is recognized to be mistaken if at some later time t_j, the agent believes that it is not the case that the execution at t_i will result or has resulted in the desired outcome.

We will refer to the recognized mistaken beliefs, intentions and actions simply as mistaken beliefs, intentions and actions from now on.

Note that recognizing that a belief, intention or action execution is mistaken does not require the agent to explicitly represent that it was mistaken. The agent can simply notice this mismatch between the previous and current beliefs and react to it without explicitly represent that fact. The CD player example above would be such a case.

1.3.4 Example

To illustrate the above, consider the following example about an agent Alma (we will use this name to refer to a generic agent throughout).

1. Alma has the goal of getting to town at 8:00 am this Sunday.
2. She believes that the metro usually starts running at 5:00 am.
3. So, she believes that the metro will start running at 5:00 am this Sunday.
4. This leads to the adoption of the following plan:
   (a) Get to the metro station at 7:00 am on Sunday.
   (b) Go up on the platform.
   (c) Catch the first train going towards town.

Here are three alternatives in the execution of the plan:

Alternative 0

When Alma tries to get to the metro station, she finds that the gates are locked. So step 4b fails. This leads to the failure of the intention adopting plan 4. Alma may later realize that the metro starts running on 8:00 am on Sundays, so belief 3 is mistaken. Sundays are an exception to the general rule 2.

Note that the mistaken belief 3 led to the mistaken intention 4 and to the mistaken action 4b.

Alternative 1

This time, on Saturday Alma realizes that the metro starts running late on Sunday, so Alma realizes that belief 3 is mistaken and the adoption of 4 is mistaken too so that intention should be dropped. This should then lead Alma to replan for the goal.

Alternative 2

Let's say now that instead of intending to get to town at 8:00 am, Alma needs to get there at 10:00 am and so decides to get to the metro station at 9:00 am since it opens, she believes at 5:00. The reasoning is now as follows:

1. Alma has the goal of getting to town at 10:00 am this Sunday.
2. She believes that the metro usually starts running at 5:00 am.
3. So, she believes that the metro will start running at 5:00 am this Sunday.
4. This leads to the adoption of the following plan:
   (a) Get to the metro station at 9:00 am on Sunday.
   (b) Go up on the platform.
   (c) Catch the first train going towards town.

After adopting the plan 4 she then realizes that the station does not open at 5:00 am on Sunday so that belief 3 is mistaken. The intention is not mistaken this time since the plan to get to the metro station at 9:00 am and to get the train will succeed.

In this case, the mistaken belief does not lead to a mistaken intention.

1.4 Responses to mistakes

Having defined the mistakes we are interested in studying, we now turn to the appropriateness of an agent's response to mistakes. Agents should follow the general guidelines for appropriate responses to be effective in dealing with mistakes.

The first issue to consider is how to define the mistakes we want to use to study the responses of the agent. We could use the recognized mistakes described earlier, however, that would imply that an agent that does not recognize any mistake and therefore does not respond to any mistake is appropriate. This is not desirable. A more appropriate view of mistakes now is the omniscient view presented in 1.3.2. Intuitively, we would like the agent to respond to all mistakes it makes. If it does not even recognize some mistake, then it is particularly unskilled at responding to that mistake.

An appropriate response to a mistake should be one that "fixes" the mistake as soon as possible. We would like the mistake to be fixed as soon as possible so that its effects are minimized. However, it might be that the best solution to a mistake is not to take any actions immediately, but to wait a while. This sort of behavior has to be taken into account when deciding the appropriateness of the response.

The other factor is the question of fixing the mistake. We view this in terms of the goals of the agent. The actions the agent takes should be such that its goals are not affected by the mistake. The goals could be very broad ones, for example, keeping an accurate model of the world. They could also be quite narrow, like maintaining the temperature in the room between 65 and 75 degrees say. If that thermostat agent mistakenly believes that Tweety is a pig (when Tweety is a bird), and this has no effect on its ability to maintain the room temperature, then this view of the appropriateness of responses would imply that not responding to that mistake is appropriate. On the other hand, the agent with the very broad notion of its goals cannot ignore that mistake. This does seem appropriate. Agents are typically designed with specific goals in mind and these are all that matter.

A mistake will typically have many consequences. A mistaken belief can cause other mistaken belief, the adoption of mistaken plans and the execution of mistaken actions. These actions can have many other effects in the world. A response to the mistake should take into account the consequences of the mistake also so that the response is appropriate if it entails an appropriate response to the consequences of the mistake too.

Whereas beliefs can be repaired by changing the beliefs and intentions by giving up the commitment to the plan, actions cannot be as easily repaired. The exact way to repair an action or even if the action can be repaired will be highly dependent on the domain and on the goals of the agent. Actions may not be reversible so that the response to mistakes involving these actions can only minimize the cost of the mistake to the agent. Further actions may need to be taken to mitigate the effects of that mistake on other possible goals of the agent. Assume that the goal of the agent is to cross the street and it mistakenly thinks the light is green when it is red and starts crossing. Then a repair of the action can simply be to step back on the curb. If however, before it does so, it gets hit by a truck, repairing that action is likely to he harder. It might interfere with the goal of the agent to remain in good repair and so in that case, part of the repair would be to get to the workshop for some real repairs.

Intuitively, there can be a range of appropriateness of responses to mistakes. This suggests that we have a method to quantify the appropriateness to mistakes of an agent. This will allow us to better judge the performance of agents. One approach to quantifying the repairs is to compute the cost of the mistake. A repair is good to the extent that it minimizes this cost. The cost of the mistake can include the time that the mistake was not repaired and will ensure that agents that react faster will be judged better. These costs have to be provided by the domain, and can even be used by agents to decide which mistakes to attend to in priority. We do not however explore this further in this work.

From the above we can derive the following definitions of appropriateness of responses to mistakes.

1. **Response to mistaken beliefs** If the agent holds belief ϕ at t_i and $\neg\phi$ is the case at t_i, the agent responds appropriately to this mistake if 1. if it is a goal of the agent to believe the truth about ϕ it believes that $\neg\phi$ is true at t_i and 2. it responds appropriately to all the consequences of the mistake.

2. **Response to mistaken intentions** If the agent has at time t_i an intention to execute some plan at time t_j and this execution does (or will or did) not result in the goal of the agent in intending to do the plan, then it responds appropriately to this failure if 1. it finds an alternative way to satisfy the goal 2. it responds appropriately to the consequences of the mistake.

3. **Response to mistaken actions** If the agent executes an action or a plan and that does not result in the expected outcome, then the agent responds appropriately to this mistake if it 1. finds an alternative way to satisfy the goal 2. takes further action to minimize the effect of that action on the agents goals.

We note that the appropriateness of each kind of response depends on the response to the particular mistake in question as well as the response to the consequences of the mistake. This is not going to be circular and the recursion will bottom out since these consequences are part of causal chains and causal chains are assumed not to be circular.

1.5 Some properties of mistakes

We now turn to some properties of mistakes that are relevant to this work. We would like to find general domain-independent properties rather than more specific ones. These will therefore be quite general and not quite as effective as domain dependent properties of mistakes can be.

One of the first constraints we have is that an agent cannot believe a fact and simultaneously believe it is mistaken. Then we consider propagating mistakes. In alternative 1 above, we note that the realization that a belief was mistaken enabled Alma to infer that the intention was mistaken too and saved her from going to the metro station while it was closed. There seem to be some useful relations between mistakes that an agent should be able to compute. Without these, the agent might make avoidable mistakes. The ability to predict and therefore to preempt mistakes based on some other information is a very useful ability to have.

1.5.1 Consistency

It is inconsistent for an agent to believe at time t_j both that ϕ is true at t_i and that ϕ is mistaken at t_i. It is however consistent for the agent to believe at t_j that it believed ϕ at t_i and that ϕ was mistaken at t_i.

In alternative 1 above, on Saturday Alma believes 1. that the metro starts running late on Sunday 2. that her belief that it starts at 5:00 am on Sunday is mistaken and 3. that she did believe in the past that the metro starts running at 5:00 am on Sunday.

We assume that the agents we consider have perfect memories so that their record of a past belief cannot be false. An agent might believe at t_j that it believed ϕ and $\neg\phi$ at t_i. Those beliefs did cause an inconsistency at t_i but the beliefs that the agent had these inconsistent beliefs are not themselves inconsistent.

There has to be a distinction between what the agent now believes was true in the past and what the agent believed to be true in the past. This is especially important if the agent is to explicitly represent mistakes and its past beliefs. We consider that issue in more detail later.

1.5.2 Propagating mistakes to consequences

Assume that an agent holds beliefs ϕ and ψ at time t_i and that it would not have held ψ had it not been for ϕ, that is, the sole cause for the agent believing ψ is ϕ. If it turns out that ϕ is mistaken, we want to know whether ψ is necessarily mistaken too. The same question holds for intentions and actions.

In alternative 1 above, Alma reasons that part of the reason she came up with the plan to go to the metro was that the metro starts at 5:00 am on Sunday. If she had not believed that, then she would not have come up with that plan. So, when it turns out that that belief was mistaken, she concludes that the intention was mistaken too.

Alternative 2, however, shows that this is not always the case: Alma adopts the intention because of the belief that the metro starts running at 5:00am on Sunday, but it turns out that there are other reasons for adopting that plan–all that is required is to know that the metro starts running before 9:00 am, which is true. So, the intention is not mistaken even though it depends on a belief that is.

The ability to propagate mistakes seems an important one since it allows the agent to infer that other beliefs, intentions and actions are mistaken. We have seem that this is not necessary though. The solution to this is perhaps to have the agent reason further about the kind of dependency between the beliefs and determine whether it should propagate the mistake. The information required is likely to be domain-dependent meta-information and the reasoning might be long and complex. It seems prudent to propagate the mistakes by default and to use the more complex reasoning to conclude that the propagation was mistaken whenever it is. Another way to find out that the propagation was mistaken is if the same belief, intention or action is the result of a different line of reasoning that does not involve the mistaken belief.

The conclusion then is to propagate mistakes to the consequences of the mistaken belief, intention or action by default. We however keep in mind that this is only a general rule of thumb and can be proven false by other information.

1.5.3 Propagating mistakes to reasons

Just as it seems typically reasonable to propagate mistakes to the objects (beliefs, intentions or actions) that depend on the mistaken object, it may seem reasonable to propagate mistakes in the opposite direction. This constitutes a rather primitive diagnosis of the mistake. If the agent believes ϕ and also finds that ϕ is mistaken at t_i, and if ϕ non-defeasibly depends on ψ and θ, it seems reasonable to then assert that one or both of ϕ and θ are mistaken too. For instance if the agent believes that Tweety weighs one pound and that Fred weighs 10, and later finds out that Fred is lighter than Tweety, then it has to be that Tweety does not weigh one pound or that Fred does not weigh ten pounds or both are true. It is not possible to narrow the choice further using domain-independent knowledge.

If on the other hand that was a defeasible inference, then it is not clear that either ϕ or θ is mistaken. It could simply be that this is a case of an exception to the rule and the mistake should not be propagated to ϕ or θ. Say the agent believes that Tweety is a bird and that birds usually fly, but then discovers that Tweety does not fly. It can't, on this basis alone, determine that either Tweety is not a bird or that it is not true that birds usually fly. This could simply be a case of an exception to the rule.

It seems harder to make a case for propagating mistakes to the reasons for the mistaken beliefs, intentions or actions. A diagnostic approach would provide valuable information for this kind of propagation, but a default rule for this kind of propagation does not seem warranted.

1.5.4 Propagating from actions to intentions

Similarly, is an action is mistaken, it seems reasonable that the intention that led to the execution of the action was mistaken too. When the agent finds that it cannot get into the Metro station so that its action fails, that means that its intention to use that plan to get to town fails too. If it cannot get to the Metro station, it cannot take the train to town.

This is sound provided that the failure of the action implies the failure of the whole plan. If the plan can succeed even if parts of it fail, then this is not a good inference to make. Assuming that most plans do not have that property, we do propagate mistakes from actions to intentions.

1.5.5 Mistaken mistakes

We saw that if the fact that a belief is mistaken is represented in the agent, that belief can itself be mistaken. So we have first ϕ, then that ϕ is mistaken, then that it is mistaken to believe that ϕ is mistaken. Mistakes can be iterated in this way for several levels.

If ϕ is found to be mistaken, and the determination that ϕ is a mistake is the found to be mistaken, this does not imply that the agent can simply ignore the mistakes and treat them as a double negation–a mistake of a mistake is no mistake. There might be actions taken as a result of the determination that ϕ was mistaken that can't be ignored, so we can't as a general rule cancel mistakes of mistakes.

1.6 Detecting mistakes

We now turn to the tasks an agent has to take to detect mistakes and the capabilities they imply for the agent. Recall that we are considering a wide range of agents with a wide range of capabilities. So, the capabilities and representations mentioned here are not all required for all agents. The degree to which these are present in agents is generally linked to the appropriateness of these agents to deal with mistakes. A number of the points mentioned here have been discussed above in the definition of mistakes and their properties. The emphasis now is on the capabilities that an agent should have to respond appropriately to its mistakes given that it does not have an omniscient perspective on the world.

The first step in handling mistakes is detecting them. Given the definitions above, we consider how an agent could detect that and which computational properties an agent can have which would facilitate that. Once again, we focus on domain independent factors. Recall also that there is a wide range of capabilities for agents. Some agents may have all these properties and others have but few of them. This puts the agent in different places of the range of responsiveness to mistakes.

Mistaken beliefs

The definition of mistaken belief that we have adopted is:

> An agent recognizes that a belief ϕ it held at time t_i was mistaken if at a later time t_j, it believes that ϕ was not the case at t_i.

Comparing beliefs The first capability the agent needs is to be able to compare its belief as to some state of affairs at two different times. This requires an ability to compare the internal structures that stand for its beliefs, and some way to consider at the same time the new and the old belief.

These need not be sophisticated capabilities. It could simply be a comparison of two bits, alternatively, the two beliefs could be a currently derived belief and an old one that is stored in the state of the agent. The mistake is then detected as a change in the state of the agent. Even more simply, the old belief could be implicit in the algorithm of the agent, and the detection of the mistake could be a test done by the algorithm

Choosing between beliefs Related to this, the agent has to have a way to adjudicate between incompatible states. When there is a belief at t_i that ϕ and at t_j, there is the belief that $\neg\phi$ at t_i, the agent has to choose between these two. One of them at least has to be mistaken, but the agent may not know which it is in which case it may be prudent to consider both to be possibly mistaken.

Changes in the world To decide that ϕ was mistaken at t_i, it is not sufficient for the agent to believe $\neg\phi$ at t_j. It has to verify whether the world changed in the interval. If that were so, it could very well be that the truth of ϕ changed between t_i and t_j and that the agent was not mistaken about ϕ at t_i.

This behavior is common for preconditions of actions: at t_i, before the trigger is pulled, the gun is loaded, but at t_j after it has been pulled, the gun is unloaded. At t_j the agent believes that the gun is unloaded but that does not mean that it was a mistake to believe that it was loaded at t_i–the action of shooting which has as precondition that the gun was loaded made the loaded gun unloaded.

Continuing with the above example, if at t_i the agent believed that the bird the gun was aiming at could not fly, then as the shot was fired at t_j the bird flew away, the agent should reason that it was likely to have been mistaken in believing that the bird could not fly at t_i because typically, non-flying birds do not suddenly become flying birds. The agent needs to know generally what facts can change and at what rate: in general, non-flying birds do not become flying birds. This too is a default: if the non-flying bird was tied to a chair and so was considered non-flying, and shooting the gun caused the rope to break, the bird can fly away and is now a flying bird. So in this case, it was not a mistake to think that the bird was non-flying.

Propagation In addition, the agent needs to be able to propagate mistakes as mentioned above.

Summary The list of capabilities for recognizing mistakes includes:

- Recording past beliefs.
- The ability to compare beliefs.
- Choosing between beliefs that are incompatible.
- Being aware of the way the world changes.
- Propagating mistakes.

Mistaken intention

We now turn to mistaken intentions. These were described as:

> An agent recognizes that an intention it held at t_i to execute a plan at t_j is mistaken if, at some time $t_k, (t_k > t_i)$ the agent concludes that it is not the case that the execution of the plan at t_j will result or has resulted in the goal desired.

The agent will need additional capabilities to deal with mistaken intentions.

Propagating mistaken beliefs One way of detecting a mistaken intention is to propagate a mistake from a belief to the intention. This gives the agent the possibility to realize that a plan is going to fail before it even attempts it. If the beliefs an intention depends on are false, then it is likely, but not guaranteed, that execution of the plan will not give the desired consequences. Therefore the intention to execute the plan is mistaken. For instance, in the above example, assume the agent has the plan to get to the metro station at 7:00 am and to take the train, and before it starts the plan, it realizes that the metro station will only open at 8:00 am before it starts the plan. Then it can infer that the plan is going to fail and therefore will avoid going to the metro station at 7:00 am.

Propagating mistaken actions Another way of detecting that an intention is mistaken is to have an action of that plan fail. If we propagate this failure, we can infer that the intention was mistaken. This too is not necessarily the case but is likely to be typically true.

This in turn assumes that the agent has the ability to observe mistaken actions.

Summary In addition to the capabilities mentioned above, the agent will need to

- Propagate mistaken beliefs to intentions.

- Propagate mistaken actions to intentions.

Mistaken action

An action is mistaken if the execution of the action fails. This requires the agent to observe that the action failed. This can be directly observed, or it could have to be inferred by observing some other fact in the world. If the bird flies away once it is shot at, the action of killing the bird failed because dead birds don't fly.

Capabilities needed for detecting mistakes:

Therefore the capabilities required of the agent to detect mistakes are:

- Recording past beliefs.
- Comparing beliefs.
- Choosing between beliefs that are incompatible.
- Being aware of the way the world changes.
- Propagating mistaken beliefs to other beliefs and intentions and propagating mistaken actions to intentions.
- Observing events in the world, including the effects of its own actions.

1.7 Representation

Dealing with mistakes requires the agent to have a flexible representation. This flexibility includes changing the KB with new information that is observed, withdrawing beliefs and reasoning about its own past beliefs and intentions. This requires that the agent have the ability to reflect its reasoning process. The more complete this reflection, the more flexible the reasoning of the agent is likely to be.

We list some representational properties that are desirable for the agent to respond appropriately to mistakes. The agent's representation need not have all of these properties. The properties the agent does have will influence the appropriateness of the responses of the agent to mistakes.

- **Time situatedness** allows for the following capabilities:
  - Keeping track of time as it passes. This is important in cases that the world changes. The agent needs to maintain its KB in step with the world and infer which beliefs were mistaken based on when they were held and when the world changed.
  - Accepting input from the world in time. Observations can be an important means for detecting mistakes and this depends on new inputs coming into the KB as soon as they are detected. If the agent ignores or does not pay enough attention to some observations that indicate a mistake, it will not deal appropriately with that mistake.
  - Knowing that it believed ϕ at time t_i. The agent needs to maintain a history of its beliefs for it to be able to realize that it was mistaken in the past.

  - Knowing the state of the world at any time in the past. This is not to be confused with the history of the beliefs of the agent. The true state of the world is from an external omniscient point of view and the agent can only approximate that through its beliefs. The past state of the world becomes different from its past beliefs when the agent takes these past beliefs to be mistaken. While the agent cannot be mistaken about its past beliefs, it may never know what the state of the world was and may be mistaken about that.

- **Inconsistency tolerance and meta-reasoning** allows for the following:
  - Tolerating ϕ and $\neg\phi$ and adjudicating between them. A first order system can derive all formulas from the presence of such a direct contradiction. It is then not possible to continue reasoning usefully. The agent should refrain from doing that and have a way to decide between the contradictands if possible.
  - Inspecting its reasoning to see whether there are other beliefs that were obtained from some belief. This reflects the past reasoning of the agent and is crucial for propagating mistakes.
  - Diagnosing its own reasoning. This is needed to find the cause of a mistake. This can be done by the agent reasoning about its own reasoning which it can inspect from the above.
  - Removing beliefs from its KB. First order systems do not remove formulas from the KB. This is necessary here in case of contradictions or mistakes.

- **Reasoning about actions and plans.** This is needed to the extent that the agent plans and acts in the world. It then has to reason about the consequences of actions and plans and the effect of mistakes on these.

As we have seen above, not all systems need to have the same degree of appropriateness in reacting to mistakes.

For instance, fairly appropriate systems can be produced through conditional planning [139, 45, 191, 85]. This enables the system to react with some flexibility to failures of its actions or failures of its expectations. Conditional planning requires the agent to make specific observations about the world. The future actions that the agent takes depends on these observations. The observations can be made to reduce the uncertainty in the world or to verify that the previous action of the agent succeeded. If there has been a mistake, a different path is taken in the computations. In this case the beliefs of the system are implicitly represented in the plan and the determination that there has been a mistake is made at the observation and immediately direct the system to a behavior that is intended to correct that mistake.

The conditional planning approach is economical and may be all that is needed in many cases. A problem with this approach is that the more comprehensive we try to be in detecting and responding to errors, the more complex the plans get so that it is hard to build, understand and execute them [135, 45]. One option then is not to deal with the possible failures that are judged to be relatively infrequent[134]. If the system can tolerate these occasional failures, the solution is adequate.

Logic as safety net

However, this may not be adequate if we need the agent to be autonomous and to deal with a wide range of problems, no matter their frequency. And even if all the errors considered in the planning stage are incorporated in the plan, there is always the possibility of being surprised by the world. There can be an event in the world that was never foreseen by the planner. These will cause the agent to fail.

A solution then is to use logic as a *safety net*. The more common failures and foreseeable problems can be compiled into a procedural approach, but for problems that the procedures cannot handle because they were not foreseen by the planner or because they were judged to be too infrequent to deserve resources spent on them for every execution of the plan, logic can be a good solution.

The behavior of such a system then would be that it can execute the normal behavior and deal with common problems fast. When it comes to odder problems for which there are no ready made solutions, the logic can come into play and the system will then find a solution to the problems but more slowly. This sort of behavior does not seem objectionable in people. We tend to solve routine problems fast, but more complex or infrequent problems take more time. This seems an acceptable behavior in autonomous agents also.

In the planning example above, if there are problems that are frequent and regular enough that we can predict their occurrence and have ready-made responses to them, then conditional planning seems to be a good way to deal with these failures. However when it comes to the odder and less probable errors, logic can be used to reason with and respond to them. So, despite the problems of logic, its flexibility and the possibility to describe a wide range of behaviors through logic make it a valuable component in an agent that has to respond to a wide range of mistakes.

This mixed use of logic and procedures requires these two sorts of reasoning to work together well and share information and coordinate their actions. The division between tasks done by the procedures and the logic is also an interesting problem. It need not be a fixed line for a particular agent but may change as some problems become more common and a faster response is needed. The system needs to learn how to solve these problems faster. An approach as that of [164], for instance, where logical specifications can be compiled into procedures can be useful. However this is well beyond the scope of the current work. We limit ourselves to considering logical agents and the capabilities that they will need to handle mistakes in realistic settings.

The logical approach we discuss is based in active logic. Active logic allows the kinds of representations we need: time-situatedness, meta-reasoning, inconsistency tolerance and reflection of the reasoning in the logic.

1.8 Experiments

In the next chapter we describe active logic and our implementation of it on which our experiments were based. We have built systems that detect and respond to mistakes along the lines presented above. These systems illustrate a variety of approaches to the problem of mistakes with more or less of the features mentioned included in the solutions, putting these solutions at various points in the efficiency/flexibility space.

The second part of this work concerns mistakes in dialog. Dialog is a fertile ground for work on mistakes since mistakes of various kinds abound in dialog. People are very good at detecting and repairing these mistakes before dialogs degenerate into parallel monologs. There is a need for such systems since dialog systems that are insensitive to their mistakes and to miscommunications tend to be very hard and frustrating to use. We describe some work that has been done and some work that is currently being pursued.

The third part concerns mistakes in agents. We give a preliminary account of non-monotonic reasoning seen as reasoning with mistakes in the context of our approach to mistakes. That reasoner was implemented in active logic and we discuss this implementation in some detail. A suite of test examples was gathered from the literature and these were used to test the logic. We next present the design of an agent that acts in the world and detects and repairs its own mistakes. The design takes into account the agent's mistaken beliefs, intentions and actions and attempts to represent a flexible mistake handling mechanism.

Chapter 2

Active logic and Alma/Carne

We saw earlier that being time-situated, tolerant of inconsistency, and facilitating meta-reasoning enables a system to handle mistakes in a flexible way. The system also needs to reason about actions and plans, but that can be done through the appropriate description of actions and planning and is not an inherent property of the underlying reasoning system. We need to have a system that has all three of these properties and is implementable to experiment with mistakes and build robust systems.

There are systems that do reason in each of the required ways. Situated systems are discussed, for instance in [164, 122, 44], managing inconsistencies is described in [**?**, 13, 90, 6, 15] and meta-reasoning is discussed in [11, 99, 166]. None of these systems seems to have all these properties, although some could be modified to have them. For instance, [164] discusses a situated agent and addresses some issues with time-situatedness, however, the logic and the situated automaton compiled from it do not seem to be able to access facts that were believed in the past. To do that, the automaton would have to be able to access arbitrary amounts of memory and would no longer be a simple automaton. [90] presents an interesting approach to inconsistency, however, the proposed logic does not support meta-reasoning. [11] describes a meta-reasoning executable temporal logic which gets close to what we want but does not tolerate inconsistencies. Inconsistency tolerance is hard to incorporate in a logic after it has been designed and is likely to drastically change the properties of the logic. Active logic [48], however is a logic that was designed with inconsistencies in mind and which has all three properties and has been implemented. It is therefore well-suited for reasoning about mistakes. All the implementations presented here have been based on active logic.

In this chapter we give an account of active logic and Alma/Carne which is our implementation of active logic. We first sketch the formalism of active logic and some of its properties and applications. We next discuss the design of Alma/Carne. Then we consider how mistakes are detected and handled in Alma/Carne. Following that we describe the main aspects of the implementation of Alma/Carne and illustrate that with some samples of its behavior.

2.1 Active logic

Active logic [48] was developed as a means of combining the best of two worlds – inference and reactivity – without giving up much of either. This requires a special evolving-during-inference model of time. The motivations for this were twofold: all physically realizable agents must deal with resource limitations the world presents, including time limitations; and people in particular have limited memories [8] and processing speeds, so that inference goes step by step rather than instantaneously as in many theoretical models of rationality. A consequence of such a resource-limited approach is that agents are not (even weakly) omniscient: there is no one moment at which an agent has acquired all (even all its own possible) logical consequences of its beliefs. This not only must be so for real agents (and hence for humans) but it also is an advantage when the agent has contradictory beliefs. In this case, an omniscient and logically complete reasoner is by definition swamped with all sentences of its language as beliefs, with no way to distinguish safe subsets to work with. By contrast, active logics, like human reasoners, have at any time only a finite belief set, and can reason about their present and past beliefs, forming new beliefs (and possibly giving up old ones) as they do so; and this occurs even when their beliefs may be inconsistent. (See [124] for details.)

Active logics can be seen either as formalisms per se, or as inference engines that implement formalisms. This double-role aspect is not accidental: it is inherent to the conception of an active logic that it have a behavior, i.e., the notion of theoremhood depends directly on two things that are not part of traditional logics: (i) what is in the current evolving belief set, and (ii) what the current evolving time is.

2.1.1 Formalism

Just as there are several first order logics depending on the set of predicate, constant and function symbols, so there are several active logics. The formal changes to a first order logic required here are, in some respects, quite modest. The language can be that of a first-order logic, perhaps augmented with names for expressions to facilitate meta-reasoning.[1]

The principal change is that inference rules become time-sensitive. The most obvious case is that of reasoning about time itself, as in the rule

```
i:              Now(i)
               ----------
i+1:            Now(i+1)
```

The above indicates that from the belief (at time i) that the current time is in fact i, one concludes that it *now* is the later time $i + 1$. That is, time does not stand still as one reasons.

add inheritance+ ?

Temporal logics [3, 114, 163] also have a notion of past, present and future, but these do not change as theorems are derived. These are specification logics external to the reasoner. This contrasts strongly with the agent-based on board character of active logic. Executable temporal logics do have a sense of time, see 2.6.1.

Technically, an active logic consists of a first-order language, a set of time-sensitive inference rules, and an observation-function that specifies an environment in which the logic “runs”. Thus an active logic is not pure formalism but is a hybrid of formal system and embedded inference engine, where the formal behavior is tied to the environment via the observations and the internal monitoring of time-passage (see [48] for a detailed description). Further formal details are shown below.

All rules of inference in the active logic are applied to all the formulas in the knowledgebase (KB) at every step to generate the formulas for the next step. Therefore an active logic is a forward chaining reasoner which gradually derives (ideally) all the consequences of the initial set of formulas in the KB.

2.1.2 Properties of active logic

Active logics are able to react to incoming information while reasoning is ongoing, blending new inputs into its inferences without having to start up a new theorem-proving effort. So external observations of actions or events can

[1]Time-sensitive meta-reasoning is the key to deadline-planning, contradiction-repair, and most of the features distinguishing active logics from other logics. (This meshes well with work in human metacognition; see for instance [131].) Collaborative systems must reason not only about their own beliefs and actions but also about the beliefs and actions of their partners. This is a yet more involved kind of meta-reasoning, studied both in traditional formalisms and in active logics.

be made during the reasoning process and also factored into that process.

Thus the notion of theorem for active logics is a bit different from that of more traditional logics, in several respects:

1. **Time sensitivity.** Theorems come and go; that is, a wff once proved remains proved but only in the sense of its being a historical fact that was once proved. That historical fact is recorded for potential use, but the wff itself need not continue to be available for use in future inferences; it might not even be reprovable, if the "axioms" (belief) set has changed sufficiently. As a trivial example, suppose $Now(noon) \rightarrow Lunchtime$ is an axiom. At time t=noon, Now(noon) will be inferred from the rule given earlier, and Lunchtime will be inferred a step later. But then Now(noon+1) is inferred, and Lunchtime is no longer inferable since its premise Now(noon) is no longer in the belief set. Lunchtime will remain in the belief set until it is no longer "inherited"; the rules for inheritance are themselves inference rules. One such involves contradiction; see next item.

2. **Contradictions.** If a direct contradiction (P and $\neg P$) occurs in the belief set at time t, that fact is noted at time t+1 by means of the inference rule

```
t:                  P , ~P
               ------------------
t+1:            Contra(t+1,P, ~P)
```

   See [125] for details on handling contradictions.

   Truth maintenance systems [43] also tolerate contradictions and resolve them typically using justification information. This happens in a separate process which runs while the reasoning engine is waiting. Although justification information is important for dealing with contradictions and their consequences, we do not think that this will work in general. The reasoning needed to resolve the contradiction will depend, typically, on the very information that generated it. Resolution of contradictions is itself a reasoning process much like any other and cannot be isolated from the logic that generated the contradiction.

3. **Metareasoning**

   In active logic, there is a single stream of reasoning, which can monitor itself by looking backwards at one moment to see what it has been doing in the past, including the very recent past. All of this is carried out in the same inferential process, without the need for level upon level of meta-reasoners. This is not to say that there is no metareasoning here, but rather that it is "in-line" metareasoning, all at one level. The advantages of this are (i) simplicity of design, (ii) no infinite regress, and (iii) no reasoning time at higher levels unaccounted for at lower levels. A potential disadvantage is the possibility of vicious self-reference. This matter is a topic of current investigation. However the contradiction handling capability should be a powerful tool even there.

2.1.3 Applications of active logic

Active logic has been applied to solve a wide range of problems. In all of these examples, active logic has been used to model the behavior of an agent. Executing the logic generates the behavior desired in real-time.

The three wise men

In [47] active logic was used to solve the three wise men problem [107]. Elgot-Drapkin describes the problem as follows:

> A king wishes to know whether his three advisors are as wise as they claim to be. Three chairs are lined up, all facing the same direction, with one behind the other. The wise men are instructed to sit down. The

> wise man in the back (wise man #3) can see the backs of the other two men. The man in the middle (wise man #2) can only see the one wise man in front of him (wise man #1); and the wise man in front (wise man #1) can see neither wise man #3 nor wise man #2. The king informs the wise men that he has three cards, all of which are either black or white, at least one of which is white. He places one card, face up, behind each of the three wise men, explaining that each wise man must determine the color of his own card. Each wise man must announce the color of his own card as soon as he knows what it is. (The first to correctly announce the color of his own card will be aptly rewarded.) All know that this will happen. The room is silent; then, after several minutes, wise man #1 says "My card is white!".

The problem was to reproduce the reasoning of wise man # 1. To solve the problem, wise man #1 needs to reason about the reasoning of the other wise men over time. He needs to infer that since the wise men are wise, if one of them had the white card, they would have deduced it in some short interval. Since they don't say anything within that interval, it has to be that wise man #1 himself who has the white cars. The solution requires the logic, simulating the reasoning of wise man #1, to represent the reasoning of the other wise men, and to represent that enough time had passed for the other wise men to have discovered the solution if they could.

Nell and Dudley

In [133] active logic was used to model an agent planning and executing a plan to achieve a goal by a deadline while time is passing. The problem to be solved is the Nell and Dudley problem [112]: Nell is tied to the rails and a train is bearing down on her. How can Dudley save her? The logic is meant to model the reasoning of Dudley. Dudley must form a plan to save Nell. But he can't take too long to do so or she will be killed. He can't wait to complete the plan before he starts to act otherwise it will be too late. He has to interleave planning and acting while keeping an eye on the time left. Nirkhe et al specify an active logic that implements this reasoning.

Natural language processing

In [76, 141] active logic has been used to solve some problems in language understanding caused by false presuppositions and implicatures. The tolerance for contradiction and meta-reasoning of active logic were essential for that work. In [184], active logic is used to implement a dialog manager in a conversational system. Active logic facilitated the dialog processing in case of the system misunderstanding the intentions of the user. More detail on these is provided later.

2.2 Implementation choices and features

One can implement an active logic for each problem one wishes to solve. This is how active logic had been used for the problems mentioned above. However, implementing a new logic for each application is not effective. While individual problems may require some special representations and procedures, all of the active logic solutions share a common set of characteristics.

Alma/Carne is our implementation of active logic meant to provide the common core representational and reasoning services for active logic applications. Alma (Active Logic MAchine) is the logical reasoning engine while Carne is a separate process used to run procedures and to provide an interface to external processes.

An active logic system is specified by making assertions into Alma in the Alma language and by optionally providing specialized representations and procedures in Alma and in Carne. This specialized knowledge allows the active logic to be tailored to the specific needs of the problem to be solved. For example, in [76] we needed representations of context that are not conveniently expressed in the Alma language. The procedures can be used to do reasoning that is not best done logically.

In this section we describe the Alma language and the choices made in designing Alma/Carne. A later section gives more detail about the implementation and execution of Alma/Carne.

2.2.1 The Alma language

The Alma language is similar to a first order language with a number of reserved predicates and the rules for constants, variables and formulas are reminiscent of Prolog. The details are available in the Alma/Carne manual in Appendix XXX. A brief account of the language and of some of the reserved predicated is given below.

Predicate constants are represented by strings that start with a lowercase letter. Variables start with an uppercase letter. Logical operations are represented in prefix form and each sentence is followed by a period. Unless explicitly quantified, variables are taken to be universally quantified.

The following examples of Alma formulas and the corresponding first order logic representations can give an idea of the language:

`p(a).`	$P(A)$
`p(X).`	$\forall x\ P(x)$
`if(p(X), q(X)).`	$\forall x\ P(x) \rightarrow Q(x)$
`forall(X, if(p(X), q(X))).`	$\forall x\ P(x) \rightarrow Q(x)$
`if(and(p(X), q(X)), r(X)).`	$\forall x\ P(x) \wedge Q(x) \rightarrow R(x)$

2.2.2 Reasoning issues

The main reasoning issues to be decided relate to the choice of inference system for Alma and the implementation of inheritance of formulas from one step to the next.

Resolution based

The main rule of inference used in Alma is resolution. While this does not result in a complete reasoner in forward chaining, Alma allows backward chaining formulas which gives completeness for first order logic. The advantage of resolution is that there is no need to choose which rule of inference to apply, the disadvantage, conversely, is that there is less control than if one used natural deduction rules, for instance. This also implies that the formulas in the Alma language are represented in Alma in conjunctive normal form. They are however not limited to Horn clauses.

Inheritance

The formulas in the KB are assumed to be inherited from one step to the next by default. The result is that there is no need to explicitly infer that the formulas are inherited from step to step as the reasoning proceeds. However, one needs to explicitly delete a formula that should not be present in subsequent steps. Since the number of formulas inherited is usually more than the number deleted at each step, this improves the efficiency of the system.

Backward search

In addition to the forward chaining proofs, Alma can also prove formulas by contradiction in a backward chaining way. A backward search for ϕ is initiated by asserting $\neg\phi$ in the KB in a special context and doing the usual forward chaining proof until a contradiction is derived. The special context results in the contradiction being handled differently from the other contradictions in the logic which results in ϕ being asserted in the KB.

2.2.3 Control issues

Logic typically brings flexibility but at the cost of efficiency. Since active logic is meant to be the on-board reasoner for an agent, the control of reasoning is an important aspect of the design. We give an account of the control algorithms currently implemented in Alma. There is however scope for much further work in this aspect of the system.

Top-level control

As with active logic, Alma runs in "steps" that are sequentially numbered. The step number is used as the time value for Alma. In each step, the rules of inference are applied to extend the derivations by at most one inference rule application. The inferences in one step are considered to occur atomically. This roughly results in an incremental breadth first forward chaining proof procedure. The KB changes with the steps with some formulas added and some deleted. A formula that is deleted at each step is the `now(t)` formula. At step `n`, the KB contains `Now(n)`, and this is deleted and `now(n`1)+ is added for the next step, step `n+1`.

New formulas only

The description of active logic implies that all inference rules are applied to all formulas at each step. This is likely to be inefficient, especially since the majority of the formulas are inherited from one step to the next. Alma ensures that each rule of inference applied involves at least one newly derived formula so that we do not repeat inferences that have already been done. This is also helpful when we want to delete a formula from the KB and don't want the formula to be rederived in a future step. Since the formulas the deleted formula had been derived from are not new, they will not produce the deleted formula later on.

This heuristic causes problems with formulas with negative introspection (see later for details). Negative introspection of ϕ is satisfied if ϕ is *not* present in the KB. Therefore removing ϕ may make it possible for formulas that depend on negative introspection of ϕ to participate in an inference. This is not available if we strictly follow the strategy above. The (not very efficient) solution used is to verify that these formulas can be used at each step.

Different implications

A reasoning and control issue is that there are three kinds of implication operations instead of the usual one. `fif` is like the usual material conditional except that it only asserts its consequent in a step when all the conjoined antecedents are satisfied at that step. The result is that `fif` formulas cannot combine with other implications to give various combinations of implications and that the `fif` formulas cannot be contraposed. They can only be used to assert their consequents.

`bif` also specifies an implication, but one that is only used in backward chaining contexts. These are therefore only used in proofs by contradiction and will not affect the usual forward chaining procedure.

`if` are the usual material conditionals that can be used both in forward and backward reasoning.

For example, if there are the following formulas in the logic:

```
if(and(p, t), s).
fif(and(p, q), r).
bif(p, u)
```

If p is added, we get `if(t, s)` but not `u` nor `fif(q, r)`. If we then add `q` so that both `p` and `q` are in the KB, we will get `r`. `u` will only be obtained if we start a backward search for it.

The agenda

Sets of formulas that can potentially be used in an inference rule are put on an agenda and then the rules are executed on these formulas. Alma allows the number of inferences considered at each step to be limited either by specifying the maximum number, or by specifying the time allowed for each step. In the latter case, the inferences are halted as soon as the time limit is exceeded.

The agenda is sorted at each step, before the inferences are attempted. The sorting algorithm is specifiable by the user. This allows for a finer degree of control that can vary according to the domain of reasoning. The sorting algorithm can be written so as to depend on the contents of the KB as well as on other properties of the KB and the formulas. This gives the user the flexibility to specify complex control procedures. Alma also allows the sorting formula to be influenced by the logic. This allows the logic to reason about and modify its own execution.

2.2.4 Meta-reasoning

Meta-reasoning is a distinguishing factor of active logic. Alma has an extensive range of predicates and procedures that give access to information about formulas and the reasoning process. These can be used in the Alma formulas and allow the reasoner to reason about its own reasoning. We describe some of these tools here.

Time

As mentioned above, time is associated to step numbers. This is available for reasoning though the predicate `now(T)`. There is just one instance of `now(T)` in the KB at any time and this changes with the step number. The simplest use of this is to do an action, for instance asserting a formula in the KB at some time (step number).

History

Alma records the changes in the KB at each step of the computation. From this list of added and deleted formulas which represents the history, Alma can compute whether some formula was in the KB at some time in the past, or which formulas were in the KB at some step. One way to access this information is through `pos_int(F, T)`. `pos_int/2` represents the relation between formulas and the times at which they were in the KB for times earlier than the current time. As computation proceeds, the domain of `pos_int` grows.

If `F` was in the KB at time `T`, this is true.

Introspection

Alma can introspect in its KB to verify whether a formula is present at the current time. An interesting application of the positive introspection is to do negative introspection. The negative introspection can be used as a time-situated approximation for non-provability. Instead of finding "is it the case that ϕ is not provable", Alma computes "is it the case that ϕ has not been derived until now". The answer to negative introspection can change with time if ϕ is derived. Negative introspection can be used as a basis for non-monotonic reasoning and is a source of mistakes. If formula `F` is in the KB now, `pos_int(F)` is true. Note that this version of `pos_int` has just one argument. To do negative introspection, we only need to negate the positive introspection.

Names

To refer to and assert properties of the formulas in its KB, all formulas in Alma have names. Names can be assigned by the user, and in this case, the user can assert properties of formulas as part of the description of the domain, for instance, that a formula is a default and that that default is preferred to another. The names can be parameterized by variables appearing in the formula which makes it easy to refer to specific instances of universally quantified formulas. Formulas are named by `named`. An example of naming a formula is `named(if(bird(X), animal(X), birdsAreAnimals(X)))`. The name of this formula is `birdsAreAnimals(X)`. If we instantiate this formula with `Joe`, the resulting name is `birdsAreAnimals_Joe)`.

Properties of formulas

In addition to the properties of formulas asserted by the user, Alma records various properties of the formulas during execution. These are available to the user for meta-logical computations. Some of the properties that Alma stores or computes when requested by the user are:

- The time (step) at which the formula was first derived, using `name_to_time`.
- The formulas from which it was derived, using `name_to_parents`.
- The formulas derived from it, using `name_to_children`.
- The derivation of the formula, using `name_to_deriv`.

2.2.5 Contradiction detection and handling

An inference rule detects direct contradictions, i.e. the presence of ϕ and $\neg\phi$ in the KB at the same time, where ϕ is a literal. This results in the contradictands and their consequences being *distrusted* and a formula `contra(N1, N2, T)` being added to the KB. `N1` and `N2` are the names of the contradictands and T is the time at which the contradiction was found. When a formula is distrusted, it cannot be used in any further inference. However, it can be inspected and reasoned about. The `contra(N1, N2, T)` assertion can be used to start reasoning about the contradiction and how the logic can resolve it.

A distrusted formula can be *reinstated* by the `reinstate` reserved predicate. This adds a new instance of the formula to the KB. It can be used in the contradiction resolution axioms when the contradiction has been resolved. Once reinstated, a formula can be used in inferences whereas the formula not reinstated remains unavailable.

2.2.6 Procedures

The ability to specify new representations and procedures conveniently facilitates the use of Alma for many problems. It is not always easy or desirable to represent complex structures as first order formulas or to represent procedures as sequences of axioms. This is especially useful if these computations do not need to be reasoned about. Alma provides facilities to represent and use special representations and procedures in both Alma and Carne. For instance, we typically are not interested in parsing natural language sentences in the logic but parsing is one of the tasks Alma may have when it is used for natural language processing, so the ability to call a parser to parse the inputs is very useful. Similarly, there may be other forms of reasoning that we may want to do but not logically. For instance, we may want to reason probabilistically about part of the domain, in this case, Alma can make use of a Bayesian net to do that computation.

Alma user-defined procedures

The user can specify new procedures and representations in Alma. Alma formulas can then access these procedures using the predicate `eval_bound`. This allows any procedure to be run in the Alma process. A number of the reserved predicates for meta-reasoning are invoked in this way. The procedures run in Alma are meant to be short quick procedures that do not unduly delay a step. If special representations are to be used, procedures to access and modify these should be provided and these can be used in the Alma formulas.

2.2.7 Carne

Carne is a process separate from Alma that communicates with it and runs procedures that would take too long to run in the Alma process. There are special predicates available in Alma that can be used to request that Carne procedures be run. The results of the computation are asserted in the Alma KB by Carne whenever they become available. Alma does not have to wait for the procedure to terminate before going on with its reasoning. A procedure started at some step in Alma may only return several steps later.

The ability of Carne to assert formulas in Alma and its independence of the Alma process allow it to be used as an interface between Alma and external systems. In this way, Alma can be embedded into complex systems as the Trains-96/Alma/Carne system (see later). Messages from the rest of the system to Alma are sent to Carne which translates them from the inter-process communication language to a form suitable for Alma and asserts those formulas in the Alma KB. Alma can send output to or query the rest of the system by sending messages to Carne which then converts them to the appropriate form and sends then to the other modules in the system.

The Carne algorithm

On startup Carne reads Prolog files that specify the computations that Alma can request and connects to Alma through sockets. It then loops waiting for inputs either from Alma on the sockets or from the outside world on its standard input. In the first case, it calls the program specified by the Alma code and updates Alma as to the progress of the program (see below). In the second case, Carne processes the input message and adds the resulting formulas to the Alma database. Carne currently has a KQML parser which converts KQML messages at the input to Alma sentences.

There are two reserved predicates that Carne provides to procedures:

- `af(P)` asserts `P` into the Alma database.
- `df(P)` deletes `P` from the Alma database.

These are used by procedures called by Ala to modify the Alma KB. They are also used by Carne to assert inputs from the outside world into Alma.

2.2.8 The Alma–Carne interface

The following predicates are available in Alma for communication with Carne:

- `do(P, I)`, when asserted in the Alma KB triggers an inference rule that sends a message to Carne to start procedure `P`, `I` is an identifier for that call.
- `doing(P, I)` is asserted in the Alma KB when Carne starts executing `P`. This and the next two formulas are asserted as a result of messages sent to Alma from Carne.
- `done(P, I)` is asserted and `doing(P, I)` is deleted when the procedure terminates successfully.
- `error(P, I)` replaces `doing(P, I)` if the procedure fails.

The `doing`, `done` and `error` predicates are useful for Alma to keep track of the status of actions it has started. An `error` predicate, in particular, signals that an action has failed. This is useful in detecting actions that fail.

2.3 Mistake handling in active logic

In this section we consider how the active logic system handles mistakes. Since this is a general logic that can reason about a wide range of domains, the mistake detection and handling capabilities are equally general and non-specific. More detailed strategies have to be provided by the domain one is reasoning about. Examples of this are given later. We intend these strategies in Alma to be general procedures that are specialized by statements in the logic provided when axiomatizing the domains of interest. This is similar to the approach to control taken in [66] but it is not clear whether this will prove to be effective. [128] provides a counterexample to that domain independent approach to control.

2.3.1 Detecting mistakes

Mistakes are detected as direct contradictions between formulas in the KB. This does not identify what the mistake is, but merely asserts that there has been a mistake. The contradictands are the immediate causes of the contradiction and one of them at least has to be mistaken. This is represented in the KB by the assertion `contra(N1, N2, t)` for a contradiction between a formula named `N1` and one named `N2` detected at time `t`. The reasoning for deciding which of the contradictands is mistaken and whether there are any other causes of the mistake has to be specified in the domain axiomatization. The meta-reasoning predicates in the logic, especially those relating to the derivation of formulas can be useful.

The above detects mistaken beliefs. The same mechanism can be used to detect mistaken actions through contradictions between the expected outcome of the action and observations or other reasoning that imply that the action failed. If we know that an action A has a consequence ϕ and after the action we notice that $\neg\phi$, we can conclude that the action failed. The expectations and the subsequent facts have to be of the form ϕ, $not(\phi)$ for the mistake to be detected. We discuss in a later chapter how that could be arranged. If the axiomatization is such that the prediction of ϕ is simply represented as ϕ in the KB at the appropriate time, and the observation that $\neg\phi$ is similarly represented, then the contradiction detection mechanism will detect a problem between these two beliefs. Once the contradiction is resolved (through domain specific axioms) the mistake can be possibly propagated to the action (see the previous chapter).

Similarly, propagation is used to detect mistaken intentions. These can be detected by propagation from beliefs that gave rise to the intention or by the failure of actions that are part of the intended plan.

2.3.2 Handling mistakes

Since Alma does not identify the mistakes, the handling of the mistakes is very general. Once `contra(N1, N2, T)` has been asserted signaling a mistake in either `N1` or in `N2`, Alma *distrusts* the possible mistakes in the KB. These consist of the contradictands and their consequences. While there might be other mistakes related to this contradiction, particularly formulas from which the contradictands are derived, the logic does not have enough information to determine which these are. Distrusting formulas prevents them from being used in further inferences–the logic cannot reason with them, but allows then to be reasoned about. This gives an opportunity for domain specific axioms to infer the identity of the mistaken formula.

Notice that both contradictands and their consequences are distrusted. Since one of the contradictands is true, it is a mistake to distrust that one and its consequences. The `reinstate` reserved predicate reinstates a distrusted formula. This can be used by the mistake handling axioms in Alma to correct that mistake when the cause of the mistake is identified. Of course, this can be mistaken too.

2.3.3 Representing mistakes

Active logic only uses the `distrust` predicate to represent possible mistakes. Other more precise representations of mistakes have to be provided by the domain axiomatization. The exact form of this representation and when it is used determines the flexibility of the system as we will see later.

2.4 Alma/Carne implementation

The reasoning system described above has been implemented and is used in a number of applications. In this section we give some details of these programs. More information can be obtained from the Alma/Carne manual (see Appendix XXX) and from [150].

2.4.1 Alma

Alma has been implemented in Quintus Prolog [174]. It can run either in stand-alone mode, or embedded in a larger system with Carne as interface to the rest of the system. Alma reads files in the Alma language that specify its behavior as well as Prolog programs that the user specifies for specialized representations and computations.

When embedded in a system, Alma is usually set to loop continuously, at each iteration computing a step. A lower bound on the delay between two steps can be set. In stand-alone mode, Alma is usually stepped through a step at a time under the control of the user, although it can be allowed to loop too. Stand-alone mode is more convenient for interactive use and development of domain axiomatizations. This is very useful since the interactions of formulas over time can sometimes be surprising.

In either case, Alma can write history and debugging information to files. The history can be used to examine the reasoning episode or to recreate it later(see below). There are several levels of debugging information available that provide more or less detail about the reasoning process. These are mainly used to debug the internal procedures of Alma.

2.4.2 Carne

Carne is also written in Quintus Prolog and is run connected to Alma through sockets. While Alma can be run without Carne, the converse is not possible. It also receives inputs from external programs. The inputs to Carne are specifications of procedures that are to be called from Alma and procedures that implement translations from external languages to the Alma language. Carne waits for inputs from either Alma or the external interface. If Alma requests a program to be run, Carne does so and asserts formulas in the Alma KB representing the status of the execution. The procedure called has predicates available that allow it to assert and delete formulas in Alma. In case input from the outside are sent to Carne, these inputs are translated and the resulting Alma formulas are asserted in the Alma KB.

2.4.3 Interface

Alma has an graphical interface written in Java that allows the user to interact with Alma more conveniently than through text input. With that interface the user can control the running of Alma, view the evolution of the KB in real time, skip back through time to see previous steps of the computation, and examine individual formulas in greater detail. These facilities make it much easier to develop domain axiomatizations and debug the behavior of Alma.

The interface can also be run from a history file of a prior reasoning episode. This is useful if one wants to carefully re-examine the behavior of Alma in a prior reasoning episode. It is also useful in illustrating the behavior of Alma. Since the behavior is defined by the history file and does not change, one can give an account of the reasoning in each step to illustrate the behavior of the logic and the details of a domain axiomatization.

2.5 Examples of behavior

In this section we illustrate basic behaviors of Alma. In each case, we show the state of the KB initially, and the state of the KB at later times to show the result of the computations. The initial state of the KB consists of the Alma formulas describing the domain together with the initial step number which is asserted by Alma. We show the additions and deletions to the KB at interesting steps instead of repeating the contents of the KB at each step. The non-monotonic reasoning section has several examples of the behavior of Alma when used together with a theory that specifies its behavior as a non-monotonic reasoner.

2.5.1 Modus ponens

The axioms entered into Alma are:

```
if(p(X), q(X)).
if(q(X), r(X)).

p(a).
q(b).
not(r(c)).
```

These formulas (whose meanings are intuitive) are added to the KB at step 1 and as the logic is subsequently run we

Step	Event
2	Add `now(2)`
2	Add `not(q(c))`
2	Add `r(b)`
2	Add `q(a)`
2	Add `if(p(X), r(X))`
2	Delete `now(1)`

get:

3	Add `now(3)`
3	Add `not(p(c))`
3	Add `r(a)`
3	Delete `now(2)`
4	Add `now(4)`
4	Delete `now(3)`

Note that

- At each step the new `now` formula is added and the old one deleted. Since the operations in one step are considered to occur atomically, we do not have in any step, both `now(t)` and `now(t`1)+.
- Alma does all one-step derivations in each step.
- After one step, at step 3, the logic derives `not(q(c))`, `r(b)` and `q(a)` by applying resolution to the appropriate formulas at step 1.
- Inference is done on all axioms so that at step 2 we get `if(p(X), r(X))` from `if(p(X), q(X))` and `if(q(X), r(X))`.
- At step 3, we get the formulas which need 2 steps to be derived from the initial KB: `r(a)` and `p(c)`.
- So, those formulas that are derived after 2 steps (`r(a), not(p(c))`) are derived at step 3 and those that need just one step are derived at step 2.
- After step 4, the only events that occur are the times changing by deletion of the previous `now` and addition of the next one.

2.5.2 Time situatedness

We illustrate a very simple example of time-situatedness. The formulas in the KB simply assert a formula in the KB at the right time.

```
fif(and(setTime(T), now(T)),
    conclusion(alarm(T))).

setTime(10).
```

We use the `now` predicate to tell when the time we are interested in has been reached. At that time we simply assert `alarm`. Note the use of `fif` here. If we had not done so, and used `if` instead, then for each time step, we would get an instance of an application of modus ponens so that at step 4 say, with `now(4)` we would derive `if(setTime(4), alarm(4))`. Most of the formulas are not useful for our purposes and would just clutter the KB.

When we run the logic, as before the formulas are added to the KB at step 1 and from then on the only formulas that change are the time until we get to step 10. Recall that `fif` will only result in a new formula when all of its antecedents are in the KB. This happens here only when `now(10)` is in the KB.

Step	Event
1	Add `now(1)`
1	Add `setTime(10)`
1	Add `fif(and(setTime(X), now(X)), alarm(X))`
2	Add `now(2)`
2	Delete `now(1)`
...	
10	Add `now(10)`
10	Delete `now(9)`
11	Add `now(11)`
11	Add `alarm(10)`
11	Delete `now(10)`

Note that the `alarm` formula is not added at time 10 but rather at time 11. At time 10, the logic notices that it is time 10. It uses this in an inference then, but the result of the inference is only added to the KB at the next step. This gives the strange effect of `alarm(10)` being added at time 11.

2.5.3 Negative introspection

Negative introspection is a quite powerful tool and can be used to implement non-monotonic reasoning as we illustrate here: if something is a bird and we don't know that it does not fly, then it does. We have two candidate birds: Tweety and Fred. We know that Fred does not fly, but have no information about Tweety.

Note that the negative introspection succeeds if positive introspection fails. This is a predicate that is computed on demand which is why it is enclosed in the `eval_bound` predicate (see the manual for details of that). Each time an inference rule is applied to a formula containing an `eval_bound` with negative introspection, that negative introspection is recomputes so that if the formula was added or deleted from the KB in the interval, that will be taken into account.

```
if(and(bird(X), eval_bound(\+ pos_int(not(flies(X))), [X])), flies(X)).

bird(tweety).
bird(fred).
not(flies(fred)).
```

We omit the initial database and time formulas and show the relevant formulas only.

Step	Event
3	Add `flies(tweety)`
4	Add `not(\+pos_int(not(flies(fred))))`

Note that the fact that Tweety flies is derived as usual but that Fred flies is not derived because of of the denial of that fact in the KB. This denial does not however stop us from deriving that Tweety flies. Note also that as a side effect of computing the `eval_bound`, its failure is asserted in the KB at step 4. The formula says that it is not the case that the logic negatively introspects that Fred does not fly, or that the logic introspected that Fred does fly.

2.5.4 Contradiction detection

We repeat the axioms in the previous example, but leave the negative introspection out. We expect then to derive a contradiction between the assertion that Fred does not fly and the derivation that Fred does fly.

```
if(bird(X), flies(X)).

bird(tweety).
bird(fred).
not(flies(fred)).
```

Step	Event
2	Add `11:not(bird(fred))`
2	Add `10:flies(fred)`
2	Add `flies(tweety)`
3	Add `contra(11, 6, 2)`
3	Add `distrusted(11, 2)`
3	Add `distrusted(6, 2)`
3	Add `distrusted(10, 2)`

Note here that `11:not(bird(fred))` expresses that the name of the formula `not(bird(fred))` is `11`, and similarly for formula `10`. We show the formula names here because it is needed to understand the subsequent formulas, but keep in mind that all formulas are named.

In this case, at step 2 the logic derives that Fred is not a bird from the facts that Fred does not fly and that birds fly. It also derives that Fred flies since it is a bird. These are formulas 11 and 10 respectively. At the next step, Alma finds that there is a contradiction between 11 (Fred is not a bird) and 6 (Fred is a bird) which was in the KB in the beginning. Then, both these formulas and their consequences are distrusted. This includes the fact that Fred flies (10) since that was derived from 6 (Fred is a bird). So there are three formulas distrusted: 11, 6 and 10. Since 10 is already distrusted, it does not produce a contradiction with the fact that Fred does not fly. This could have worked out differently if the contradiction between Fred flying and not flying had been found first. The same formulas would have been distrusted, but there would have been a different contradiction. Once these formulas are distrusted, no consequence can be derived from them. The KB does not subsequently change.

Note that the fact that Tweety flies is not affected by any of the reasoning about Fred. We do not, for instance, distrust the consequences of the default that birds usually fly even though one instance of this fails.

2.6 Related Work

The main feature of active logic is time-situatedness and this makes possible the other features of meta-reasoning, and contradiction tolerance. We consider three formalisms that have a notion of time situatedness that approaches that of active logic: Executable temporal logic, Sneps, and Golog. Neither of these have the same facility as active logic to reason with and about the current time, although executable temporal logics get the closest. Neither of these formalisms however, allow contradictions to appear and reason about them as active logic does. MML, the executable temporal logic that we consider does allow meta-reasoning but a contradictory situation is seen as a failure of the building of a model for the formulas of the logic.

While the other formalisms have some of the features of active logic, it does seem that active logic has a unique combination of features that allow it to be used in a variety of ways the other formalisms can't be used. In particular, we need the features of active logics to handle mistakes flexibly.

2.6.1 Executable temporal logic

Temporal logics [3, 14] are used to reason about changing worlds and are used for specifying and verifying dynamic systems. Executable temporal logics attempt to build models of temporal logic formulas by executing the temporal formulas.

We focus in discrete temporal logics where the formulas can refer to individual steps in the execution, and in particular to the Metatem system [57]. The approach is to see formulas as expressing what the future should be, given the past [59]. The slogan being "Declarative past and imperative future". Any formula can be rewritten in the form *Condition about past* $\rightarrow$ *Condition about the present and future* so that executing a formula requires the interpreter to verify that the antecedent was true in the past and to then constrain the future based on the consequent of the formula. The logic is monotonic in that if previous steps constrain a future state to make some formula true, there cannot be a later state that makes that false. For instance, if ϕ is asserted to be false from $t = 10$ on, we cannot assert ϕ at $t = 15$. The interpreter may have to make choices in the execution that lead it to the impossibility to satisfy some formulas. If ψ is asserted to be true for ever some time after $t = 25$ and the interpreter chooses to make it true at $t = 29$ but later finds that there is an assertion that ψ be false at $t = 35$, the choice to make ψ true at $t = 35$ needs to be undone. In that case, the system backtracks and makes alternate choices.

This approach is close to the active logic point of view in terms of the agent being situated in time. A fundamental difference is that while active logic simply produces the behavior specified, executable temporal logics attempt to build a model of the system specified by searching for histories that satisfy the constraints described. This is why we have backtracking in cases that the logic picks the wrong history to expand. The backtracking of the program in the case of failure cannot happen in active logic–we cannot undo a choice made earlier but have to move on from there. Another difference is that the monotonicity of Metatem contrasts with the nonmonotonicity of active logic.

In [11], Metatem is extended to MML and includes meta-reasoning. The domain of the logic in this case includes names of the object level formulas. Variables are divided into two sorts: the object and the meta variables. This logic allows one to use MML for meta-interpreters, and for control of inference for example.

This extension of Metatem allows the logic to refer to its own formulas but it is not clear that there are facilities to reason about the derivation or other properties of these formulas that have proved to be useful in active logic.

2.6.2 SNePS

Redo this In [86] the Sneps semantic net [172, 171] reasoning system is extended to reason in time. The approach is to use NOW as a deictic pointer to the current time. NOW is a meta-logical variable and is not a term in the language. The value that NOW takes is denoted by $*NOW$ and that varies with time. $*NOW$ denotes the current time and cannot refer to the past or the future. The time changes only when the agent acts, and actions can only be done $*NOW$.

A distinction is made between "eternal states" which represent facts that are always true and temporary states which represents facts true at some particular time. The latter are associated to time through a formula $Holds(s, t)$.

The authors identify two problems with their representation:

- There is a problem in specifying formulas of the form $whenpdoq$ where p and q are temporary states and q denotes an action to be done. $*NOW$ cannot be used in specifying that action because it is not a term in the language. The solution is to eliminate references to time in such formulas and allow the inference procedure to add the appropriate value of $*NOW$ when the rule is used.
- The second problem relates to the time that it takes to verify p in the above. If the agent has to take some action to do so, time will advance and p will be verified at a time different from the time inserted by the inference rule in the solution to the first problem above. The result is that the action is never done even though the condition is true. The solution proposed to this is to consider NOW to take values not of a time $*NOW$ but of a set of intervals of different granularities. In the example then, NOW initially represents a coarse-granularity $*NOW$, but when the action is performed to verify p, a finer granularity is used which is still within the coarse NOW so that the condition is true at that coarser granularity and the action can be performed. The details of that solution are not presented.

Active logic contrasts with this approach in that Now is a predicate in the language, and the arguments of Now are integers that represent time. Now and its values can be used in the language. This gives greater expressivity than the Sneps approach. We can easily state that an action is to be done 5 time steps after the conditions are true, for example, because we can compute with the times (steps) in the language itself. Once the condition is true, we know the current time and compute the time 5 time steps from there and assert that the action is to be done when that time comes along. The representation of NOW as a meta-term outside of the language seems to restrict the extent to which the logic can reason about its own time.

Another difference between our approach and the Sneps approach is that time in Sneps is linked to the actions the agent does. In active logic, by contrast, time is taken to increment independently of what the agent does. This seems to better situate the active logic in the real world even though we do not have at this point, a guaranteed correspondence between steps and clock time.

As far as the problems mentioned above, there is no need in active logic to specify a temporal argument for formulas as the above since the antecedents are evaluated and the consequents asserted at the same step. If the antecedent is a complex formula that needs to be evaluated at the same time, the fif form can be used. Since we do not have to instantiate the formula with the time at which it is evaluated, the problems with the Sneps approach do not come into the picture.

2.6.3 Golog

Golog [102] is a logic programming language meant for high-level programming of agents and is based on the situation calculus [109]. The primitive actions are described in terms of their preconditions and their effects. The frame problem [109] is solved [159] assuming that all the ways a predicate fluent can change are given in the action description and that there are no state constraints; that is there are no indirect actions. Given a description of the primitive actions and a program, the Golog interpreter uses theorem proving to determine whether there is a series of primitive actions satisfying that program.

Time is introduced in an extension of Golog [160]. The system is not situated in time but can reason about temporal constraints in the programs. In that sense it is different from active logic. However, if the sequence of actions is to be executed, then the robot doing so needs to obtain the current value of time before deciding what to do next. The robot is supposed to monitor the environment and recompute the rest of the schedule based on the time. Although the actions of the robot in this case depend on the time at which it is trying to execute the program, this does not have the flexibility of an active logic agent that can reason in and with time.

2.7 Future work

Alma/Carne can currently be used in a range of experiments in knowledge representation and reasoning. The framework of Alma provides flexibility for those kinds of experiments. We discuss some additional features that can make it more usable. Alma is also not very practical for non-toy applications because of efficiency concerns. This can be addressed by better control of the reasoning.

2.7.1 Other systems of inference rules

Alma uses resolution (see [28] for instance) as inference rule. This is convenient as resolution is the only rule needed and this saves us from having to choose which rule of inference to apply. However, resolution is not complete in forward chaining which is the main mode of reasoning in active logic. Although we do not want to derive all possible formulas, it would be useful to have the possibility to do so and explicitly control which inferences to make. This could be done with a different set of inference rules.

Another advantage of having an alternate set of rules is that there can be heuristic information in how people encode axioms. Although $p \rightarrow q$ and $\neg q \rightarrow \neg p$ are logically equivalent, intuitively, we would state the first form if we expect to see p more often than to see $\neg q$. This heuristic information is lost in resolution where both are expressed in the same form. This information can be used for controlling the reasoning: in the first case we would use the axiom in the presence of p and in the second in the presence of $\neg q$, for instance.

A possible alternative set of rules to be implemented in Alma is the natural deduction rules [101]. These are complete rules, are intuitive and can provide good heuristics for controlling the reasoning. The Oscar system [148], for example, is based on natural deduction. It should be possible to replace the resolution module of Alma by a natural deduction module while keeping its time-sensitivity and meta-reasoning components intact.

2.7.2 Built-in non-monotonic reasoning

Non-monotonic reasoning is essential for commonsense agents and a number of applications of active logic use this meta-reasoning feature to implement non-monotonic reasoning. It is not effective to re-implement different non-monotonic reasoning schemes in each active logic application, just as it is not efficient to re-implement an active logic for each problem we want to solve. In a later chapter we describe a non-monotonic reasoning scheme implemented in Alma. This is meant to be a general non-monotonic reasoner and is intended for any application that needs this kind of reasoning. This reasoner is implemented as an application of Alma rather than as a primitive feature. The result is that it is rather awkward to use the non-monotonic logic. Integrating this into Alma would make it more usable.

2.7.3 Improved interface

The GUI currently available with Alma is provides basic functionality. We can view the KB as it evolves, and at any state in the past and can inspect individual formulas and can to a certain extent control the logic from the GUI. Other features that need to be added are a more extensive set of control for Alma, a view of derivations, a way to modify the database by directly manipulating formulas in the GUI and making it possible to run the interface over the internet. The latter would make it possible to have an Alma server in one location and clients anywhere on the web.

2.7.4 Control of inference

Alma, by default, computes its derivations in a forward chaining breadth-first way. If left to itself, such a policy can result in larger and larger numbers of inferences at each step, though that also depends on the formulas in the KB and on the inputs. In the Trains-96/Alma/Carne application, the logic is run for thousands of steps without this sort of explosion in the number of inferences. The rapid increase in possible inferences can slow down the reasoner considerably. And it is likely that a majority of the inferences made are either irrelevant to the computation we are interested in or are different derivations for the same result. This is a major bottleneck in using Alma in larger, more interesting domains when the axioms may not be as carefully written as in the Trains-96/Alma/Carne application.

There are two approaches to this problem currently implemented: the possibility of backward search and the possibility to control the inferences done. Backward search is very useful if we know exactly what we want to prove. In many instances though, we depend on the logic to derive formulas that will be relevant to our interests, without having a good idea about the formulas initially. This sort of reasoning is well suited for forward chaining as Alma currently does.

The control framework provided in Alma seems adequate: we can reorder the list of inferences that are potentially going to succeed, and we can limit the number of inferences done in each step. The problem is to find strategies that will help make the logic more effective without ruling out making some inferences. There are a number of known techniques for choosing clauses for resolution, for instance unit preference, set of support and so on [165]. These should be implemented as in Alma and the user given a choice to apply them, and additionally, we ought to find more heuristics that may be better suited for our computations.

Several approaches to controlling inference have been proposed and can be useful in this case. First of all there is the question as to whether it is worth to have control knowledge [10, 61] and whether control knowledge should be domain independent or specific [66, 128, 187]. Approaches include meta-reasoning [21, 31, 36, 190, 77, 20], decision theory [37, 175, 83] and relevance reasoning [12, 180, 181, 103, 178]. Some of this work could be adapted to Alma and implemented as logical axioms that Alma would use to control its own reasoning.

Part II

Mistakes in dialog

Chapter 3

Mistakes in dialog–an introduction

Natural language communication makes miscommunication inevitable. Miscommunication can be seen as a mistake in the conversation process and in dialog, this typically has to be detected and repaired promptly for the dialog not to degenerate into separate monologs. In this part, we discuss some experiments in detecting and repairing mistakes in dialog processing.

We start with a discussion of the conversational adequacy hypothesis which states that an agent is conversationally adequate iff it can handle miscommunications in a reasonable way even if it has a minimum knowledge of the language in which the dialog is being conducted. This implies that dealing with miscommunications and mistakes in general, is a central part of natural language understanding.

We next describe the ongoing Trains/Alma/Carne project which aims to enhance the conversational adequacy of the Trains-95 system by replacing the dialog manager in that system with one based on Alma/Carne and which can therefore reason in time and with contradictions. The result is that some errors that are not handled in the original system can now be handled appropriately.

In the other chapters in this part, we discuss work done on mistakes in presupposition and implicature processing. Implicatures and presuppositions are crucial components of natural language processing and dealing with errors in these is therefore essential for conversations to proceed smoothly. Our work there focuses on aspects of deriving the meaning of utterances, oriented more towards pragmatics of language than towards the semantics or syntax. We assume that basic logical forms for the utterances are available. In our work, we simulated that by entering the appropriate logical sentences for the algorithms to work on. In a complete system, these would be generated by a parser and semantic processing module. In both cases, our work relates the utterance we consider to the larger context of the conversation. In the presupposition case it is to resolve conflicts between the context and the presuppositions of the utterance, and in the implicature case, is is to resolve conflicts between implicatures of the utterance and the context.

3.1 The pervasiveness of miscommunication

Human dialog is riddled with miscommunication. We continually make mistakes in conversation or even in reading text. These are detected either on our own or with the help of the conversational participant. The mistake can then be corrected and the dialog proceeds. In this section we provide some evidence for the pervasiveness of miscommunication and some approaches to it.

3.1.1 Young children

Very young children seem to have the ability to deal with mistakes even before they have a good grasp of a language. In fact the ability to detect mistakes seems to help them learn the language. Clark [32] discusses some behaviors in young children that fall into that category.

Monitoring one's ongoing utterance An example of this was seen in a 2 year, 6 month child practicing parts of speech (in this case its pronunciation of "berries") on its own:

> Back please / berries / *not* barries / barries, barries / *not* barries / berries / ba ba.

Here the child, even though it is not proficient in language is nonetheless able to monitor its own performance and notice that it has made mistakes. The mistakes of interest are mispronunciations. Once it detects a mistake, it corrects itself and continues practicing. Detecting and responding to mistakes, at least in this simple way, seems prior to language processing being fully developed.

Checking the result of an utterance Children at least as young as 5 years, 4 months comment on and correct the utterances of others. They also verify that the listener has understood their utterances and attempt a repair otherwise.

This active search for possible miscommunications and the effort to correct them seems to require a rather sophisticated model of the communication process which is available to quite young children.

Predicting the consequences of using inflections, words, phrases or sentences, including judging the politeness of utterances, which is exhibited by children aged four and a half. Children can also correct word order in sentences judged "silly". Clark cites instances of this being done by two-year olds.

Here too, there is a facility to detect mistakes and respond to them early in the child's development.

3.1.2 Map-task corpus

The map-task is an experimental setting where two people each have a map that represents the same place, but where neither of the maps are complete and the maps are not identical to each other. The map of one of the participants might have a feature that the other map does not and neither person knows what the other's map looks like. One of the participants is the *giver* and has a path on his map. The giver gives instructions to the *follower* so that the follower reproduces the path on his map. The instructions have to be given orally and are made difficult by the lack of a shares view of the map. The participants have to build that common view of the map from their statements as they solve the problem. We are now only interested in the miscommunications which occur very frequently in this task, for instance the giver might refer to some feature on his map that the follower does not have, A more complex situation is when the giver has a rock say, on his map and refer to that and the follower too has a rock on his map, but these are in different locations. The error need not be immediately noticed and has to be discovered later. We illustrate this process with a few samples of conversation from the corpus.

This somewhat artificial scenario is replete with miscommunications. We present a few examples of them here.

Example 1

[G] So, fg—eh, go to the left two inches.
[F] fg—Eh, right, okay.
[G] No, left.

Here the giver (G) needs the follower to go left and the follower says "right", maybe to acknowledge the request. This is interpreted by the giver as the follower moving right and therefore as a mistake–the giver intended for the follower to move left but concludes (perhaps mistakenly) that the follower moves right instead. The giver then attempts to correct the follower. This miscommunication is resolved later in the conversation over several exchanges.

Example 2

[G] You'll ... You'll go north ... and then you'll turn west,
onto the bridge.
[F] Okay.
[G] Or, east. Correction.
[F] Okay.

In this case, the giver seems to initially believe that the follower should go west. But there is then an apparent change in mind and the giver decides that the follower should go east instead. So the belief of the giver that the follower should go west is mistaken. A consequence of that belief of the giver was to tell the follower to go west in the first utterance. This is mistaken too and has to be repaired. This is done in the third utterance where the giver emphasizes that "west" was a mistake.

In these cases, it is apparent that there is a mistake committed, the mistake is noticed and a repair is done. If any of these steps does not occur, it is unlikely that the participants will succeed in the task which they do. If there had not been the possibility of detecting and repairing mistakes, it seem that this task would have been hard to accomplish.

3.1.3 Strategic competence

We now turn to a more theoretical view of miscommunication. Canale and Swain [26, 27] distinguished several competences that make up communication competence in addition to the linguistic competence identified by Chomsky [30]. Of interest to us is strategic competence.

Strategic competence is the set of strategies that are put to use when communication fails. These are of two main types: grammatical strategies that are used when grammatical competence fails, and socio-linguistic strategies that are used in situations when the socio-linguistic competence is inadequate. Some of the strategies mentioned in [182] are: approximation, circumlocution, repetition, emphasis, asking for help, miming, avoiding the problematic concepts, and abandoning an utterance already initiated.

These existence of these strategies is very interesting. First of all, it suggests that communication failure is common enough for people to have developed strategies to cope with them. Second, these failures have to be detected in the course of communication. Third, the identification of strategies to repair these failures implies that there are enough of these failures that fit into a small number of categories to make general strategies useful.

Strategic competence is useful in various circumstances like in the early stages of second language learning [26]. Savignon [169] notes that communicative competence can be present in the absence of grammatical or discourse competence. But when strategic and socio-linguistic competence are present, one can indeed communicate (non-verbally) provided there is a cooperative interlocutor. She further points out that

> The inclusion of strategic competence as a component of communicative competence at all levels is important because it demonstrates that regardless of experience and level of proficiency one never knows *all* a language.

If one never knows all the language, one is bound to commit errors in communication and therefore the need for strategies to detect and repair the errors.

An example of the use of strategic competence from [169] is shown in Figure 3.1. In this example, breakdown in

In a crowded New York deli a visiting Frenchman has ordered a Swiss cheese sandwich.

Waitress:	What kind of bread do you want for your sandwich, white, whole wheat or rye?
Frenchman:	(Wh)ye.
Waitress:	White?
Frenchman:	*(Wh)ye.*
Waitress:	*White?*
Frenchman:	Whole wheat.

Figure 3.1: A linguistic deficiency resulting in message abandonment.

T1	Mother:	Do you know who's going to that meeting?
T2	Russ:	Who?
T3	Mother:	I don't know.
T4	Russ:	Oh. Probably Mrs. McOwen and probably Mrs. Cadry and some of the teachers.

Figure 3.2: Russ notices an inconsistency at T3 and makes a repair at T4

communication is caused by differences in pronunciation. The Frenchman detects the problem and tries to solve it using repetition, then emphasis, and finally by abandoning the message altogether. This also illustrates the negotiation of meaning involved in the use of strategic competence as noted in [182].

Tarone also includes as a necessary criterion for the use of strategic competence the requirement that the speaker be aware that the linguistic structure needed to convey his meaning is not available to him or to the hearer. This leads to the use of the strategies to help get the meaning across. We note that this requires the speaker to recognize and reason about its own limitations and capabilities and its adequacy for various tasks. This is a sort of meta-reasoning that the speaker needs to undertake.

This suggests that the speaker has a model of the communication process and is aware of the shortcomings of his own or the hearer's linguistic computations. This is analogous to our notion of a reasoner having a model of his own reasoning. If we look at the problem of getting one's meaning across as some sort of reasoning problem that needs to be solved, then the speaker needs to have a model of the reasoning process in general and to know what capabilities are available to the speaker or to the hearer, and to reason whether these are adequate.

3.1.4 Failed expectations

McRoy and Hirst [118, 120] consider misunderstanding and repairs in dialog. In their model of conversation, misunderstanding is signaled by an inconsistency between the expectations of a dialog participant and an utterance. The agent must then reason about and explain this inconsistency. This can lead to a change in the interpretation of previous parts of the dialog and trigger a repair utterance. McRoy discusses Figure 3.2 where T3 is inconsistent with the expectations of Russ and is repaired in T4.

An intuitive account of this episode is as follows. Russ believes that Mother knows who will be at the meaning and wants to tell him. So, Russ responds with T2. T3 provides evidence that Mother does not in fact know who will be there. Russ's belief that Mother knows who will be at the meeting is mistaken and so was his response at T2. Russ then provides a more appropriate response at T4.

This approach to mistakes is close to our approach of the response to a mistaken action. The speaker performs some communicative action or commits to some intention with some expectation as to the trajectory of the world in the future. But if the observations contradict the expectation, a mistake is inferred and a repair is attempted.

3.2 The conversational adequacy hypothesis

From examples like the above, [142, 143] identifies two different competences that a conversational agent needs to have to converse: object-level competences and miscommunication competence. The miscommunication competence is seen as essential for adequate conversation and can facilitate conversation in cases in which the participants do not have a good command of the language.

3.2.1 Object-level competence

Object-level competence are those competences that an agent needs to have conversations in which there are no mistakes. These competences include

- Grammatical competence. This is knowledge of the structure and vocabulary of the language.
- Object-level inference. This is the ability to reason about the objects mentioned in the conversation, but not to the conversation itself or to the mental objects of the agent.

3.2.2 Miscommunication competence

To be able to handle miscommunications, the agent needs to have miscommunication competence. The components of miscommunication competence include

- Situated time. The ability for the agent to represent and reason about the time in which it is reasoning and conversing. This is to be contrasted to temporal logics where the agent reasons about time that is not the time in which it is itself reasoning.
- History. The agent needs to have a record of events that happened. These include utterances and other external events as well as mental events by the reasoner.
- Linguistic objects. The agent needs to be able to treat words in the utterances as objects that can be reasoned about and distinguish between the words and their meaning.
- Contradiction. Since miscommunication can result in inconsistencies in the KB, the agent needs to tolerate contradictions and have some means to control its effects.

Note that the capabilities are essentially meta-reasoning capabilities and are present in active logic and in Alma/Carne. Therefore Alma/Carne provides a base upon which to experiment with miscommunication competence.

3.2.3 Conversational adequacy

An agent is conversationally adequate if it can hold an “adequate” conversation with other agents. A conversationally adequate agent has to have a *free-ranging* ability whereby it can converse in a reasonable way about any topic. It is not necessary for the agent to know all topics, but it should be able to realize it does not know about the topic of the conversation and then try to ask questions and learn. In that case, there can still be an exchange of information, but of a more limited kind. The need for information exchange rules out agents that talk past their interlocutor, pursuing their own interests without taking into account the needs of the interlocutor. This frequently happens in current automated conversation systems.

Conversational adequacy is something no conversational software seems to have. They seem to have limited understanding of what is said to them in a narrow topic and once the conversation moves outside those narrow confines, the performance tends to degrade catastrophically.

Intuitively, a conversationally adequate program will be one that one can converse with and not give up in frustration at the opaqueness and stupidity of the agent as with many current systems.

3.2.4 The conversational adequacy hypothesis

The hypothesis put forth in [143] is that

(i) SUFFICIENCY: as long as there is at least a *weak* ability in the object capacities (inference, learning and language) then effective conversation can proceed if there is a *strong* miscommunication competence.

(ii) NECESSITY: no matter how strong the object capacities, effective conversation cannot proceed if there is not a strong miscommunication competence.

This hypothesis is not provable because of the lack of a clear definition of conversational adequacy, it can be operationalized through Turing-test like tests and gives a direction for work in better conversational agents.

3.2.5 Examples

We have earlier come across some examples that support this hypothesis:

- The Frenchman. In the example above, the Frenchman does not have a good command of English. He does not eventually get what he initially wanted but both he and the deli employee realize that there is a breakdown in communication and try to repair it. Despite their efforts, the Frenchman does not get his meaning across. In the end, he perhaps reasons that the cost of getting across his meaning was not worth the trouble and settled for white bread. However there was a recognition on both sides that there was miscommunication and a diagnosis of the problem. The resources available did not seem sufficient to solve the problem however. Although the Frenchmen had weak linguistic competence in English, he had good miscommunication competence.
- Children. In the examples, the child does not have good competence in vocabulary–it is still learning the words. However, it does know enough to notice that it has made a mistake and uses that to improve its vocabulary. The recognition of the mistakes and their subsequent correction seems essential for learning the language and seems to be present (to a degree at least) in very young children who have not learned a language yet. Here too, linguistic competence seems to be lacking, but miscommunication competence is not. The difference here is that this happens in a very young child which seems to emphasize the importance of miscommunication competence. In fact it seems that without something like miscommunication competence, the child would not be able to recognize its own mistakes and that would make learning the language harder.
- The map-task. In the map-task, the conversational participants do have a good command of the language, but the fact that they do not have a common view of the map makes communication difficult. The conversants recognize this through miscommunications. This gives them the opportunity to build and correct that common view as the conversation proceeds. The experiment is successful if the participants share enough of the same view for the path built by the follower to be the same as that of the giver. Without sensitivity to miscommunication and repair it would not be possible to complete the task.

3.3 Trains/Alma/Carne dialog manager

One way of providing evidence for the conversational adequacy hypothesis is to compare the adequacy of a system without (or with little) miscommunication competence with one with some (or more) miscommunication competence. This is the motivation for the Trains/Alma/Carne dialog manager project, described in [184, 4].

The Trains-96 [55] dialog manager was replaced by one implemented in Alma that was designed to have better miscommunication competence. The resulting system handled a class of mistakes in conversation better than the original system.

The dialog in the Trains system concerns moving trains to and from various cities. Our main interest lies in resolving potential miscommunications in the dialog. Active logic is well suited for the ongoing inference needed in dialog management. The metareasoning capabilities facilitate reasoning about breakdowns in the conversation.

3.3.1 Alma/Carne as dialog manager

A user of the system types in requests that are parsed by the Trains parser and sent to Carne which translates the parse and asserts it as a set of formulas in Alma. Alma uses these formulas to determine the intention of the user. This process may need more information from the domain. These requests are sent to Carne which in turn queries the Trains system using the appropriate syntax for the Trains system. The response from the system is translated into Alma formulas that Carne asserts in Alma. This may be repeated several times. Once the intention is determined, Alma sends, through Carne, instructions to the system for it to accomplish the tasks required.

The current focus is on resolving errors in reference resolution in the dialog manager. The user may intend to refer to some train using a linguistic expression (“The Washington train”) that can be interpreted in several ways (Metroliner or Acela). If the dialog manager chooses an interpretation (Metroliner) other than what the user meant (Acela), the response is likely to be erroneous. Feedback from the user (“No”) helps the dialog manager recognize its error and resolving it results in the user and the system agreeing on the meaning of the expression (Acela). The system needs to realize that its choice of referent for the expression was wrong and change it.

3.3.2 The levels of representation

We briefly describe the representation used in the Trains/Alma/Carne project. More detail on the representation and the algorithms used can be found in [184]. The utterances of the user are taken to be requests from the user to take actions in the system. The information contained in the utterance is represented in several distinct layers:

L-req	This represents the words said.
I-req	This represents the direct logical translation of the utterance with all the ambiguities present.
D-req	This is similar to the I-req but with the ambiguities removed.
P-act	This specifies how the agent will satisfy the request in the D-req.
E-act	This is the actions the system actually takes to satisfy the request.
O-act	This is the observation by the system as to whether or not the action was successful. This may not always be present.

These levels are meant to represent the different stages in the interpretation and execution of the request of the user. One of the advantages of retaining them explicitly in the KB rather than letting these come and go as the computation proceeds is that it simplifies dealing with miscommunications.

3.3.3 Errors at each level

Miscommunication can occur at each of the levels of the representation. We take miscommunication to be mistakes. Examples of errors at each level are:

L-req A mistake at this level can be obtained when the system gets its input from a speech recognizer. Then the user may say “Boston” but the system recognizes that as “Ballston”.

I-req Errors at that level can be caused by the parser failing to obtain the correct structure for the utterance, for instance.

D-req The example presented above about a mistake in the reference resolution falls in this category.

P-act Here we have mistaken intentions. This could happen if, for instance the user wants to send a train to New York and the plan chosen to do so is mistaken because of fire on the tracks which prevents the plan from being successful.

E-act This is a case of mistaken action, for example, the action of ending the train from Washington to Baltimore fails because of fire at the Baltimore station.

O-Act A failure at this level indicates that an action or an intention has failed. For instance the action of sending the train from Washington to Baltimore might result in the train being stranded between the two cities.

The representation of the separate pieces of information used to derive the intention of the user and the response to that intention can facilitate reasoning about and responding to mistakes. Without these representations, it would be hard for the system to know where exactly in the processing of the utterance the mistake first occurred which makes repairs harder.

3.3.4 An example

To illustrate that a system is more conversationally adequate if it can tolerate contradictions, does meta-reasoning and is sensitive to passing time, the Trains/Alma/Carne system was compared to the Trains-96 system. For the situation in which

- There are several trains a some city X.
- The user requests “the X train” to be sent to city Y.

The following behavior is produced:

- The system chooses one of the trains at X as “the X train” and sends this to Y.
- This was not what was meant by the user though, and he rejects the move by saying “No”.
- The system returns the train to the original position

Both systems behave similarly to that point. But if the user repeats the request to send “the X train” to Y, the Trains-96 system sends the same train it did the first time. The Trains/Alma system however, chooses one of the other trains to send since the user had rejected the train first sent.

In both cases, the system realizes that its action was mistaken and undoes it. However, the Trains-96 system does not seem to realize that its choice of referent for “the X train” was mistaken and repeats this choice. This response maybe adequate if the user then uses a different description for the train it needs to send, but not in the general case. The Trains/Alma/Carne system does realize that the first object that it resolved “the X train” to is not appropriate and corrects the choice. If the second choice is rejected too, the Trains/Alma/Carne system will pick another train. If all the trains have been selected and rejected by the user, it requests clarification from the user.

3.4 Presuppositions and implicatures

The rest of this part discusses in some detail detecting and responding to mistakes in presupposition and implicature processing. The ability to presuppose facts and to come to implicatures of utterances is essential for a natural behavior in conversation. Mistakes will be present in these cases and these mistakes have to be handled. This is done following our general approach to mistakes, specialized to handle just the cases of interest.

Chapter 4

Presuppositions

As a conversation proceeds, the new sentences of the participants is interpreted in the context of what has been mentioned earlier. The *discourse context* represents, for a participant in a dialog, what is being conveyed in the dialog. We expect the context to grow as the conversation proceeds and more information is available. The problem we consider here is a problematic aspect of the relationship between the discourse context and the presuppositions of a new utterance. See [75, 76] for details.

The presuppositions of an utterance are pragmatic propositions that have to be assumed for the utterance to be understood. The problem of computing these presuppositions has a long history and is not yet fully resolved. Heretofore one of the most highly regarded treatments was that of Heim whose approach to presupposition projection [78] gives intuitive results in many instances. However, there are cases where the presupposition is inconsistent with the context and where Heim's approach fails. The approach we propose here perfoms similarly to Heim's in the unproblematic cases, but when there are inconsistencies, we take more care in resolving the contradictions and the results of our algorithm accords better with intuition than Heim's.

4.1 Presuppositions

Presuppositions are a class of propositions that have to be assumed for a natural language utterance to make sense. For instance, if someone says "The roses are red", one has to presuppose that there are some roses. Without that assumption, there is no referent for "the roses" and the truth of the sentence cannot be evaluated. The roses could have been mentioned earlier and added to the discourse context, or if this is the first time the roses are mentioned, the presupposition that there are roses can be added to the context.

Presuppositions are a pervasive part of natural language and can be triggered by many constructions, for example

- Definite noun phrases as above. "The roses are red" presupposes that there are roses.
- Cleft sentences as in "It is the roses that are red" This presupposes that there are roses.
- Possessive noun phrases as "John's roses are red" This presupposes that there are roses that belong to John.
- Aspectual verbs as "It has stopped raining" This presupposes that it was raining.
- Factive verbs as "John regrets picking the roses" This presupposes that John picked the roses.

Note that presuppositions cannot be deduced from the utterance, they must be assumed before the meaning of the utterance can be determined. The sentence constructions above generate *potential* presuppositions that may or may not eventually be part of the context. For instance, “The roses are not red” generated the potential presupposition that there are roses. If this is then followed by “Because there are no roses”, that presupposition should no longer be part of the discourse context, in fact it contradicts the utterance.

4.1.1 Updating the context

There are three possibilities when we have a new utterance: (1) the context entails the presuppositions of the utterance; (2) the context entails the negation of the presuppositions; (3) the context entails neither the presupposition nor its negation. Cases (1) and (3) seem relatively unproblematic. In (1), there is no special problem relating to presuppositions. In (3), we can *accommodate* the utterance by adding the presupposition to the context (assuming we know that the context does not entail the negation of the presupposition). We consider case (2) in more detail.

Heim’s rules

Rules for updating context have been proposed by Heim[79, 78]. A function $+$ maps a context C and an utterance U onto a new context. There is also an accommodation mechanism that handles cases when the presupposition is not entailed by the context. The four rules that specify $+$ are listed below. The propositions are taken to be sets of possible worlds. The proposition corresponding to an utterance U is denoted by $[[U]]$.

CCPB

$$C + U = C \cap [[U]]$$

This is the basis case where the utterance is represented by an atomic proposition. Updating the context simply intersects the possible worlds of the context with those of the utterance. This results in possible worlds that satisfy both the old context and the utterance.

CCPA

$$C + (U and V) = ((C + U) + V)$$

In the case of a conjunction, we first intersect the first proposition with the context, followed by the second one.

CCPN

$$C + (not U) = C \setminus (C + U)$$

In the case of negations, we first “imagine” what the set of possible worlds would be like if the proposition were true, $C + U$, and remove those worlds from the current context. Or, we remove from the old context the worlds that are consistent with the utterance.

CCPC

$$C + (if\ U\ then\ V) = C \setminus ((C + U) \setminus ((C + U) + V))$$

Conditionals “if U then V” are treated as “not(U) or V” which is equivalent to “not(U and not(V))”.

The accommodation process is described by the accommodation rule. Let $Pr(U)$ be the presupposition of utterance U. Then if the context does not entail the presupposition of the utterance, we first add the presupposition then add the utterance: $C + U = (C + Pr(U)) + U$. The accommodation also occurs if the context entails the negation of the presupposition as in case (2) above. In that case, $(C + Pr(U))$ ends up being an empty set of worlds which indicates a contradiction. This can cause problems in some cases.

4.2 The problem

Consider the following:
U1: There are no roses.
U2: So the roses are not in the fridge.

We take the logical from of U1 to be $\neg\exists x\ Roses(x)$[1] and that of U2 including the presuppositions that there is a fridge and that there are roses to be $\neg(\exists x\ y\ Roses(x) \wedge Fridge(y) \wedge In(x,y))$. Let C_1 be the context initially, then applying Heim's rule to these utterances we get:

1. We start with context C_1.
2. After U1, the new context is $C_2 = C_1 + [[\neg\exists x\ Roses(x)]]$
3. Applying CCPN and accommodation to U2 results in context $C_3 = C_2 \setminus (((C_2 + Roses(x)) + Fridge(y)) + In(x,y))$ Since C_2 entails that there are no roses, the proposition to the right of the $\setminus$ is the empty set of worlds and we are left with the C_2 which says that there are no roses.

C_2, at the end of $U2$ therefore ignores the contents of $U2$. In particular, the presupposition that there is a fridge is lost. Intuitively though, at the end of $U2$ we should still think that there is a fridge. This points to a problem with Heim's approach.

4.3 Our solution

We propose processing the information in a finer-grained way. After $U1$, the context contains the assertion that there are roses. The presupposition that there are roses (from $U2$) conflicts with that. It is either a mistake to believe that there are no roses or to presuppose that there are. We prefer the results of the utterance rather than the presupposition and use that to regain consistency. We then proceed to process the rest of $u2$ in the repaired context. This results in the context being accommodated with the existence of a fridge.

Heim's solution seems coarser-grained in that it rejects all the content of $U2$ once there is a problem in it without analyzing the cause of the problem and possible repairs. Our solution depends on the ability of the logic to tolerate contradictions, resolve them and proceed with the reasoning. This capability provided by active logic.

4.3.1 Representation

This solution was implemented in active logic. (This was a specialized implementation of active logic–the Alma/Carne implementation was developed later.) We first give an account of the representations used to model this problem.

4.3.2 Predicates used.

1. *now(t)* indicates that we are now at the tth step of computation.
2. *ctxt(c, t)* represents that the context at time *t* consists of the list of formulas *c*.
3. *ut('X', t)* represents that *X* has been uttered at time *t*.

[1]Note that our solution assumes that the logical forms of the utterance are available from some other module. In this project, we did not implement these other modules, but simulated them by entering their expected output into our procedures.

4. *parse(X, t)* is the parse obtained at time *t* by processing an utterance at the previous step.
5. *dfnt(X)* represents a definite description in the utterance. (We assume that this is produced by the parser).
6. *update(X, t)* represents at time *t*, elements of the discourse that still need to be incorporated into the context according to Heim's rules. X is a list of contexts, atoms from the inputs and the + and \ operators. In the subsequent presentation of rules and active logic steps, + will be denoted as PLUS, and \ as SLASH.
7. *presup(X)* marks *X* as a presupposition.
8. *exists(x, P(x))* indicates that an object with property *P* exists in the discourse context.[2]
9. *assert(X)* marks *X* as having been asserted in an utterance.
10. *contra(X, Y, t)* indicates that there is a contradiction between the formulas *X* and *Y* in the context at time $t - 1$.
11. *NULL(X)* indicates that formula X is not to be "trusted".
12. *SUSPECT(X)* indicates that formula X has given rise to a contradiction.

4.3.3 Rules of inference used.

The rules will be presented in the form:

i: X
i+1: Y

If X is believed at step i, then Y is added to the beliefs at step i+1. Nothing else is added to the beliefs that is not mentioned by these rules.

1. i: ut('X', i)
   i+1: parse(Y, i+1)

   where Y is a parse of X. This rule invokes a parser on X to get Y.
2. i: ctxt(C, i) parse(X, i)
   i+1: update(Z, i+1)

   Z is a list of operators and operands such that successively applying the operators to their operands results in updating the context with the parsed input utterance according to Heim's rules (CCPA, CCPN, CCPC).
3. i: update(X, i)
   i+1: update(Y, i+1)

   where Y is the result of applying the first operator (PLUS or SLASH) in the list Y to its arguments. There are several cases depending on the operator and on the form of the operands. For instance + adds a formula to the context.
4. i: update(X, i)
   i+1: ctxt(X, i+1)

   this rule is a subcase of the previous and is applied when all context updating is complete for one particular utterance, i.e. there are no operators in X. Once the update is complete, the new context is put back into the set of beliefs of the system.

[2]Our use of "exists" here is not the usual logical use with narrow scope. Rather, it has wider scope as used in DRT [87] and by Heim.

5. i: ctxt([…, foo(X), …, bar(not(Y)), …], i)
   i+1: ctxt([…, SUSPECT(foo(X), …, SUSPECT(bar(not(Y))), …, contra(foo(X), bar(not(Y)))], i+1)

   This rule detects direct contradictions in the context. Here, X and Y are unifiable and *foo* and *bar* are either *assert* or *presup*. Note that *foo(X)* and *bar(not(X))* are tagged as being "suspect" at *i+1*.

6. i: ctxt([…, SUSPECT(foo(X), …, SUSPECT(bar(not(Y))), …, contra(foo(X), bar(not(Y)))], i+1)
   i+1: ctxt(Z, i+1)

   Z is the context resulting from resolving the contradiction flagged at step i. The contradiction can be resolved by using various additional sources of information including:[3]

   - Other elements in the context, for example rhetorical relations, formulas in the context relevant to the contradictands, the sequence of
     inferences leading to the derivation of the contradictands.
   - General knowledge which may be outside the context (though we do not treat this here).
   - The status of the contradictands— whether they are assertions, presuppositions or distrusted.

   Resolving the contradiction can result in one or both of the formulas being distrusted, and in further changes in the context. Note that the resolution of a contradiction is itself defeasible—this resolution could later lead to other contradictions which could undo the changes done at this point.

7. i: ctxt(X, i)
   i+1: ctxt(X, i+1)

   We simply inherit the context to the next step if there is no change.

8. i: ctxt(X, i), ctxt(Y, i)
   i+1: ctxt(X ∪ Y, i+1)

   If two contexts are present at a step, perhaps an original context and one obtained by incorporating a sentence into it, we merge the two. This is a merge of formulas, not of possible worlds as before. Merging the 2 contexts could introduce contradictions in the total context. That will be detected at the next step.

9. i: now(i)
   i+1: now(i+1)

   This is the "clock rule". Time does not stand still while we are reasoning.

4.4 Steps Galore

We consider only discourses that depend on 'but', 'so', 'because', overtly. We will treat them as inter-sentential relevance markers. The steps are shown with the step number on the left hand side, next to the contents of the KB at that step.

4.4.1 The first example

We now present our first example.[4]

D_1 = ⟨There are roses and tulips. But the roses are not yellow⟩

[3] See Miller [123] for more on contradiction resolution in active logic

[4] Some details are not shown, for example the argument representing time in the predicates.

Step
0 ctxt([], 0),ut(’There are roses and tulips’)

Let c_1 = [].[5]

1 c_1, parse(and(exists(x,R(x)),exists(y,T(y))))

This is the result of parsing the utterance and inheriting the previous context.

2 c_1, update([c_1,exists(x,R(x)),PLUS,exists(y,T(y)),PLUS])

update is the result of applying Heim’s rules recursively to the parsed utterance. Note that this is in postfix form, to facilitate computation.

3 c_1, update([c_2,exists(y,T(y)),PLUS])

where c_2 = [assert(exists(x,R(x)))]

The first operation is (c_1,exists(x,R(x)),PLUS). We just assert the new atom into the context.

4 c_1,update(c_3)

where $c_3 = c_2 \cup$ assert(exists(y,T(y)))

We assert the second part of the utterance into the context too.

$\vdots$

7 c_3

At the end of processing the first utterance, the context contains the assertions that there are both roses and tulips in the discourse context. We now add the next utterance.

8 c_3, ut(’But the roses are not yellow’)

9 c_3, parse(and(but,not(and(dfnt(R(z)),Y(z)))))

The new utterance has been parsed and we now need to incorporate it into the context.

10 c_3, update(c_3,but,PLUS,c_3,but,PLUS,dfnt(R(z)),PLUS,Y(z), PLUS, SLASH])

$\vdots$

[5]We will use c_i for both the list of formulas in the context and for the predicate ctxt(c_i, j). Which is meant will be evident from the context.

12 $\quad$ c_3,update(c_4,c_4,dfnt(R(z)),PLUS,Y(z),PLUS,SLASH])

where $c_4 = c_3 \cup$ assert(but)

Now we have to add “the roses” to the context. We search in the context for roses that were previously mentioned the closest to the present time—i.e., a mention of roses closest to the tail of the list.

13 $\quad$ c_3,update(c_4,c_5,Y(z),PLUS,SLASH)

where $c_5 = c_4 \cup$ assert(x=z)

We have in fact mentioned roses before, and we make the new mention of roses designate the same roses as the previous mention by asserting x=z.

14 $\quad$ c_3, update(c_4, c_6,SLASH)

where $c_6 = c_5 \cup$ Y(z)

Set difference between the two contexts is done by adding to the first context the negation of the elements in the second context but not in the first.

Here we have a choice of what to negate: either that x=z or that z are yellow, or both. It is at this point that we make appeal to rhetorical information in *but* to help us make the best choice. We choose to negate that Z are yellow only and:

15 $\quad$ c_3, update($c_4 \cup$ not(and(assert(x=z),assert(Y(z))))

We eventually obtain:

ctxt([assert(exists(x,R(x))),assert(exists(y,T(y))), assert(but),not(and(assert(x=z),assert(Y(z))),
assert(not(Y(z))),assert(x=z).

We could of course have made a bad choice here. Had that happened, it could have led to a contradiction later on in the discourse and the choice we made at this point would then be questioned.

4.4.2 A second example

We now show an example of active logic using the rules and predicates discussed above to a garden path sentence. This is essentially D_7.

$D_9 = \langle$John bought flowers, [Are the roses in the fridge?], No, (the roses are not in the fridge), Because there were no roses$\rangle$

Here we have a case where something is first added to the discourse context only to be promptly removed. As presented in [78], the CCP rules cannot deal with this discourse. The last sentence would simply produce a null set of possible worlds, making anything that followed ”felicitous.”

To save space we will consider only the shortened discourse D_7:

$D_7 = \langle$The roses are not in the fridge. Because there are no roses.$\rangle$

Step
0 ctxt([],0) ut(’The roses are not in the fridge’)

Let c_1 = [].

1 c_1, parse(not(and(dfnt(R(x)),dfnt(F(y)),in(x, y))))

This is the result of parsing the utterance and inheriting the previous context.

2 c_1, update([c_1, c_1,dfnt(R(x)),PLUS,dfnt(F(y)), PLUS, in(x,y), PLUS, SLASH])

We get the update predicate by applying Heim’s rules as before.

3 c_1, update([c_2,c_2,dfnt(F(y)),PLUS,in(x,y),PLUS,SLASH])

where c_2 = [presup(exists(x,R(x)))]

The first operation is (c_2, dfnt(F(y)), PLUS). Since we have a definite descriptor, we first search the previous context (c_1) for a previous mention of roses. As there is none, we accomodate (globally) the context with the presupposition that there are roses.

4 c_1, update([c_3, c_3,in(x,y),PLUS,SLASH])

where c_3= [presup(exists(x,R(x))), presup(exists(y,F(y)))]

Similarly, we accomodate by adding the presupposition that there is a fridge to our context.

5 c_1, update([c_3, $c_3 \cup$ [assert(in(x,y))],SLASH]

We simply assert that the roses are in the fridge. Note that here, only one context is being updated. We do not add the new assertion globally.

6 c_1, update([$c_3 \cup$ [not(assert(in(x,y)))]])

$\vdots$

11 c_4, ut(’Because there are no roses’)

where c_4= $c_3 \cup$ [not(assert(in(x,y)))] = [presup(exists(x,R(x))), presup(exists(y,F(y))) assert(not(in(x, y)))])

After some processing, we end up with a new context that contains the presuppositions that there are roses and a fridge and that the roses are in the fridge. To this new context, the second utterance is added.

12 c_4,parse(and(because,not(exists(z,R(z)))))

We repeat the processing we did above.

13 c_4, update(c_4,because,PLUS,c_4,because,PLUS,exists(Z,R(z)),PLUS,SLASH)

⋮

17 c_4, update(c_5)

where $c_5 = c_4 \cup$ [assert(because), not(assert(exists(z,R(z))))]

We have asserted "because" in the context because "because" can serve as a clue to picking the right choice among several alternatives we could encounter in later processing.

⋮

21 ctxt([presup(exists(x,R(x))),presup(y,F(y)),asssert(not(in(x,y))),assert(because),assert(not(exists(z,R(z))))

We now have a context which presupposes that there are roses and which asserts that there are none.

22 ctxt([SUSPECT(exists(x,R(x))),presup(y,F(y)), asssert(not(in(x,y))),assert(because), SUSPECT(not(exists(z,R(z))))
contra(presup(exists(x,R(x))),assert(not(exists(z,R(z)))))]

That contradiction is detected and flagged. The formulas that caused the contradiction appear at this step flagged as being "suspect".

23 ctxt([NULL(exists(x,R(x))),presup(y,F(y)), asssert(not(in(x,y))),assert(because), assert(not(exists(z,R(z))))

Using the fact that one of the contradictands was a presupposition and the other an assertion we conclude that we were mistaken about the presupposition. This is not always the right decision but it is a good heuristic. We make the presupposition that roses exist to be suspect and we reinstate the assertion that roses do not exist (in the discourse context).

24 ctxt([NULL(exists(x,R(x))),presup(y,F(y)), asssert(NULL(in(x,y))),assert(because), assert(not(exists(z,R(z))))

Since we doubt the existence of roses, we doubt the truthfulness of the statement that the roses are in the fridge.

At the end of processing D_7, we "know" the following:

- There is a fridge.
- There are no roses.

And we have doubts about the following:

- There are roses.
- The roses are in the fridge.

And this is what we expect to get.

4.5 Discussion of the solution

In this section we highlight the aspects of the solution that enable us to reason about the mistake more flexibly than in the solution of Heim. This example does not involve actions or intentions. The mistake we deal with is a mistake in beliefs. This models a passive agent listening to a stream of utterances and incorporating the information into its KB.

4.5.1 Detecting and representing the mistake

A mistake ocurs in this problem when the context entails ϕ but the utterance being processed has $\neg\phi$ as a potential presupposition. We consider here the case that ϕ is explicitly contained in the context. If ϕ is entailed but not contained in the context, the mistake will be noticed later and the processing will then proceed in a similar way.

The assertions in the context are represented as $assert(\phi)$ and the presuppositions as $presup(\neg\phi)$. Detecting the mistake then involves detecting such a pair of formulas in the current context. This is done by rule 5. That rule removes the contradictory formulas from the context, asserts that they are $SUSPECT$ and asserts that there has been a contradiction. The fact that there has been a mistake is represented here by the contradiction assertion: $contra(assert(\phi), presup(\neg\phi), t)$. This does not identify the mistake, only indicates that one of the two formulas has to be mistaken. An example of the application of rule 5 is seen in step 22 of the second example.

4.5.2 Responding to the mistake

In addition to asserting that there has been a contradiction (mistake), rule 5 also removes the contradictory formulas from the context and asserts that they are $SUSPECT$. The removal of the formulas is the first line of defence: since one of the two formulas is false, by removing them, we minimize the derivation of false formulas.

Resolution of the contradiction is done by rule 6. This uses a preference assertion made in the KB to decide which of the contradictands to reinstate to the context at the next step. This results in $assert(X)$ formulas being added to the next step in preference to $presup(X)$ formulas. This is generally a good heuristic since the presuppositions are potential presuppositions that have to agree with the context.

Once the mistake is resolved, the status of the preferred belief, assert(X), reverts to being believed from being $SUSPECT$ while that of the mistaken belief, presup(Y), goes to $NULL$ which indicates that it is mistaken. The consequences of the mistaken belief are also changed to $NULL$.

This resolves the problem and the computation can proceed with the rest of the utterance being included into the context. Only the mistaken presupposition is removed.

4.5.3 No actions

It is to be noted that this example does not involve any actions other than manipulating the KB. Further, the changes done to the KB as a result of a mistake are simple and are the same for all possible mistakes that we consider. These factors make it easy to resolve the problem simply with the representation that we have a mistake and invoking a standard method to resolve it. The solution did not explicitly indicate which of the contradictands was mistaken even though that was discovered in the resolution of the contradiction. Instead, the repair of the context was done immediately.

If there were actions undertaken in the world or if the response to the mistakes were more complex, it is unlikely that just asserting $contra$ would have sufficed. It may have been necessary to explicitly indicate which beliefs, intentions and actions were mistaken to be able to respond appropriately. This explicit approach would also be useful in case the correction of the mistake was later found ot be itself mistaken.

4.5.4 Why does it work and Heim's does not?

The reason our approach gives better results intuitively than Heim's approach is that we do more careful reasoning about the mistake that occurs. Once an inconsistency appears in Heim's approach, the whole segment of the dialog that is involved is discarded. In our case, then we notice the inconsistency, we note that as a mistake and we find the cause of the mistake and resolve it which allows us to continue processing the rest of the utterance.

In Heim's approach, the algorithm does not "realize" that there is a mistake. There is no special datastructure or procedure that comes into play when the mistake occurs. This precludes reasoning about the contradiction as our solution does and makes it difficult to have anything but the simplest response to mistakes.

4.6 Related work

There are numerous theories of presupposition, accommodation, and the projection of presupposition. There are fewer computational implementations. And of these most do not discuss or attempt to treat the cases of actual cancellation as happens in D_7 and D_9. We have chosen to study and adopt Heim's theory because it covers many of the problematic cases and it also suggests the kind of step by step, forward chaining reasoning of active logic. Ours is an approach appealing to nonmonotonic reasoning. Other nonmonotonic approaches to presupposition include those of Mercer [121], Marcu and Hirst [105], and McRoy and Hirst [119, 120].

4.6.1 Mercer

Mercer employs a system of default rules to model the presuppositions arising from syntactic forms that appear in utterances. In [121] he deals with adverbial presuppositions such as the following:
If John kicked the ball, then Bill kicked the ball too.
If Fred called yesterday, then he will call again today.
In these cases the adverbs "too" and "again" give rise to potential presuppositions; that someone else kicked the ball and that Fred called before. But in each case the potential presupposition does not project. The examples we have been discussing are mostly cases of existential presupposition triggered by definite descriptions. We do not think that this is an important difference from Mercer's examples for the phenomena under study and we believe that we could in the future bring adverbial and other sources of presupposition into our system. The important similarity between Mercer's paper and ours is the concern with the complexity of presupposition. Now Heim's CCP rules which we implement are intended to account for projection in if/then sentences in a well–founded, uniform way. Therefore we expect that our system can deal properly with Mercer's examples. A major difference between Mercer's approach and ours is that he does not address the time evolving positing and cancellation of presupposition. This is a constant theme in the comparison of our approach with others.

4.6.2 Marcu and Hirst

Marcu and Hirst [105] present a system designed to handle cancellation of presuppositions. But they take an approach quite different from our approach. They do not model the step by step incremental reasoning about context. Rather they compute an entire new theory after each utterance. Although we have not verified this, their system may be able to get the correct results for most if not all of our examples. It appears that they would deal with a discourse like D_9 by first computing the two presuppositions after u_1. Then, after u_2 they would discard all beliefs and compute a fresh set of beliefs consistent with the entire discourse. They also develop an ontology based on Meinong's theory of objects. They use this ontology to deal with discourses about fictional entities and discourses that involve presupposition. We believe, along with others [63, 78, 88, 177, 89] that presupposition can be treated separately from fictional discourse and that we can achieve this without a Meinongian ontology. The ultimate success of our approach would bear out this claim.

4.6.3 McRoy and Hirst

McRoy and Hirst [119, 120] present an abductive treatment of misunderstanding in dialogs. By way of contrast we use a largely deductive (though time–situated) inference engine. As McRoy and Hirst note, a deductive approach leads to contradictory beliefs and the need for belief revision. However, in our approach, belief revision is handled as part–and–parcel of the inference process; it does not require an additional module or phase of processing [6]. Moreover, contrary to [82], we do not need to assume there are no "abnormalities"; or rather any abnormality is easily retracted later in the dialog when new evidence is heard.

Thus our approach is an exploration of the utility of largely deductive methods in natural language processing; when contradictions arise, our logic engine applies the applicable rules. As shown in our output traces above and in Miller [123], active logic engines are often able to reason quite effectively with contradictions. It is that fact that provides the underlying framework that we are exploiting.

Ballim and Wilks [9] provide another treatment of belief and inference, essentially context-based, that perhaps could be marshalled in similar ways to our use of active logics here. However, their treatment does not appear to be contradiction-tolerant, and their use of time is much less explicit than in active logics, and in particular the reasoning done with "viewpoints" (as their contexts are called) does not reflect a notion of current evolving time as in active logics. Instead they would apparently utilize a back-and-forth juggling of viewpoints to keep contradictions from surfacing within the same viewpoint.

4.7 Conclusion

This problem illustrates the usefulness of explicitly representing and reasoning with mistakes. Heim's appproach fails to give an intuitive solution in the cases we consider because once a mistake occurs, there is no analysis of the mistake or of its causes. Instead, Heim's procedure ignores the problematic utterance and processes the dialog as though nothing happened. In our approach, the fact that there has been a mistake is flagged by the $contra$ assertion and the logic reasons about it to resolve the problem. The mistake is localized and removed and the rest of the sentence is processed in the context of this resolution.

Our solution does not explicitly represent which beliefs were mistaken and does not explicitly represent the actions taken subsequently to repair the mistakes. We only represent that there has been a mistake and the resolution of the mistake happens procedurally. This is sufficient for this simple case, but may not be adequate in case the choice we made to resolve the mistake was found to be mistaken or if there were some external actions involved. In these cases, an explicit identification of the mistaken beleif and the actions taken would be necessary to repair the KB.

This illustrates that point that the most explicit and complex mistake handling algorithms are not always necessary. The approach we chose to repairing mistakes can be more closely matched to the needs of the problems we are addressing.

[6] Traditionally such an additional module might be treated as a truth maintenance system [43]

Chapter 5

Implicatures

We present another application of mistakes to language processing. The phenomenon we model here is the cancellation of implicatures. Implicatures occur very frequently in language processing and dialog, and sometimes one makes an implicature only to find out later on that is was a mistake to do so. We tend to smoothly adopt the correct view and continue with the dialog. This is the phenomenon we attempt to model here.

The basic theory accounting for this behavior is Grice's [72, 73, 71] theory of meaning and implicature. Although Grice's theory has been known for a long time, there have not been any commonly accepted computational implementations of it. Our algorithms that will implement a small but rather problematic part of the theory of implicatures. These were possible through the contradiction handling and meta-reasoning features of active logic.

5.1 Implicature

Implicatures, roughly, are propositions that are suggested by an utterance without being entailed by that utterance. If someone says "John is in the yard or at work." an implicature would be that the speaker does not know exactly where John is. This proposition is not entailed by what the speaker said, but seems to follow from it if we assume that the speaker is being cooperative.

5.1.1 The cooperative principle

This sort of reasonableness assumption is stated in Grice's [72] *Cooperative Principle*:

> Make your conversational contribution such as is required, at the stage at which it occurs, by the accepted purpose or direction of the talk exchange in which you are engaged.

This is assumed to be the basis upon which people converse. If we did not conform to such a principle, conversations could degenerate into a string of non-sequiturs with no useful information exchange.

5.1.2 The maxims

The Cooperative Principle is rather vague and Grice fleshes out the Cooperative Principle into a set of maxims that fall into 4 categories:

Quantity	– Make your contribution as informative as is required.
	– Do not make your contribution more informative than is required.
Quality	– Try to make your contribution one that is true.
	– Do not say what you believe is false
	– Do not say that for which you lack adequate evidence
Relation	– Be relevant.
Manner	– Avoid obscurity of expression.
	– Avoid ambiguity.
	– Be brief.
	– Be orderly.

In cooperative conversation, it is reasonable for the participants to follow these maxims and we expect them to do so. But that need not always be the case. A conversational participant can:

- Violate a maxim, in which case he may mislead the hearer.
- Opt out of the Cooperative Principle, and make it clear that he is not going to cooperate in the conversation.
- Be faced with a clash, when different maxims require conflicting behavior.
- Flout the maxim, in which case the speaker blatantly violates the maxim.

The last item typically gives rise to conversational implicature. The hearer must reconcile what the person said with the assumption that the speaker is observing the Cooperative Principle.

5.1.3 Conversational implicature

Grice then says what a conversational implicature is:

> A man who, by (in, when) saying (or making as if to say) that p has implicated that q, may be said to have conversationally implicated that q provided that (1) he is presumed to be observing the conversational maxims, or at least the cooperation principle; (2) the supposition that he is aware that, or thinks that, q is required in order to make his saying or making as if to say p (or doing so in those terms) consistent with the presumption; (3) the speaker thinks (and would expect the hearer to think that the speaker thinks) that it is within the competence of the hearer to work out, or grasp intuitively, that the supposition mentioned in (2) is required.

So if we are told “John is at work or is in the yard” in response to a query about the location of John, the interlocutor seems to violate the first maxim of quality because the assertion is less informative than is required. However, if that is all the speaker knows, this apparent violation of a maxim is a flouting and the hearer should be able to infer that the speaker does not know where exactly John is. This sort of reasoning is reminiscent of non-monotonic reasoning with the assumption that the speaker is cooperative[1].

[1]We do not however model all of this behavior in the current work.

5.2 The problem

Our main concern here is to model the making and withdrawal of implicatures in real time as a conversation proceeds. As mentioned above, implicatures are not entailed by the utterance, and so it is possible that one makes an implicature based on the assumption the speaker is following the Cooperative Principle and later find that that implicature has to be withdrawn when information is available if we find that the speaker is not following the Cooperative Principle. For instance, if one asks whether it rained last night, the response might be "The grass is wet . . ." from which one derives the implicature that it did rain, but if the speaker continues with ". . . but the street is not wet.", it seems false that it did rain and the implicature has to be withdrawn. This is another instance of a mistake: the initial belief that it did rain was mistaken and we detect that mistake once we learn that the street is not wet. If the grass is wet and the street not, then it is likely that it did not rain and the sprinkler was on.

As conversation proceeds, we make many implicatures and some of them turn out to be mistaken. The KB of an agent will vary over time with the addition and removal of implicatures. It is this change in the beliefs of the agent over time that we want to model, rather than the final state of the KB after all the information has been processed.

This work is similar to the work on presuppositions discussed earlier. The utterances in the conversation invite us to make some assertions that turn out to be false. In this case though, there is no need for our solution to maintain an explicit context for the conversation.

We illustrate the problem and our solution to it with two simple examples.

5.2.1 Example 1

Consider the following conversation fragment:

(A) Kathy: **Are the roses fresh?**
(B) Bill: **They are in the fridge.**
(C) Bill: **But they're not fresh.**

We aim to model Kathy's reasoning in response to Bill's utterances. After utterance (B), Kathy concludes that the roses are fresh. If Bill is being cooperative and obeying the maxim of relevance, then the fact that the roses are in the fridge is relevant to Kathy's question about their freshness. The possible link is that fridges are typically used to keep things fresh and so Kathy can conclude that Bill wants her to conclude that the roses are indeed fresh.

However, at (C), Bill says that they are not fresh. This is not consistent with the implicature that Kathy made and since the implicature is not entailed by (B), Kathy rejects it and prefers (C) and concludes that the roses are not fresh.

Kathy's belief about the roses' freshness starts out being unknown, then true, then false. The problem is to model this change in belief as it happens.

5.2.2 Example 2

(A) Kathy: **Are the roses fresh?**
(B) Bill: **They are in the fridge.**
(D) Bill: **But they are old.**

As in the first example, (B) implicates that the roses are fresh. But then (D) seems to implicate that they are not fresh. Kathy may then decide to go with either of the implications or none of them. This seems to depend on Kathy's belief about the power of fridges to keep old roses fresh. If she has no opinion on that, then she will have no opinion on the freshness of the roses either after this dialog. A further analysis is that Bill himself does not know whether they are fresh since otherwise he would have said so to obey the maxim of quality. He is giving all the relevant information he has and is letting Kathy come to her own conclusions.

5.3 Our solution

Our solution to this problem is similar to the solution for the presupposition case as far as handling mistakes is concerned. The problem-specific processing and the representation of the information is different.

5.3.1 Representation

We describe the operators and formulas used in this logic.

Operators

The logic used here was different from the one used for the presupposition work and included a number of new operators:

- `=>` is used to specify implications.
- `&` is used for conjunction.
- `@` is used to state the time at which a formula is taken to be true. $\phi@10$, for instance, expresses that ϕ is taken to be true at step 10.
- `?` This is a unary predicate which is true if its argument is in the KB at the time that it is computed.
- `'` This is similar to "?" except that it searches for an exact match of its argument in the KB. Unification is not allowed. So that $?p(a)$ is true if $p(X)$ is in the KB (recall that X is a variable) but $`p(a)$ is not.
- `:` This is used to associate formulas to "contexts" or groupings of formulas. Formulas are grouped together according to their purpose. $bel : \phi$, for instance says that ϕ is a belief whereas $imp : \phi$ represents that ϕ is an implicature. The grouping used are:

bel	for beliefs.
imp	for implicatures.
mt	for facts about the control of computation.
qyn(ϕ)	is used to compute the significance of ϕ when this is the answer to a yes-no question.
rel(P)	for formulas that are relevant to P.
time	for time facts.
utt	for utterances.

Predicates

There are also a number of reserved predicates which we list here:

contra	`contra(p, q)` expresses that the logic has found a direct contradiction involving `p`.
eval	This is used to compute prolog programs.
inform	`inform(x, y, p)` means that agent `X` informs agent `Y` of fact `p`.
kill	This is used to disinherit a formula. If `kill(p)` is asserted in the KB, `p` will not be present at the next step unless there is a new derivation for it.
now	`now(t)` indicated that the time is now `t`.
q_yes_no	`q_yes_no(x, y, p)` represents that `x` asks `y` whether `p` is true.
respond	`repond(x, y, p, q)` means that `x` responds to `y`'s query `q` with `p`.
whoami	`whoami(x)` says that the agent we are modeling is `x`.

Some formulas

We present some formulas and their intended meaning to clarify the syntax:

- `time:now(1)` says that the current step is 1
- `bel:roses(r1)` says that the agent believes that there are roses called $r1$
- `utt: (q_yes_no(kathy, bill,fresh(r1))@1)` says that the agent believes that there was an utterance at time 1 which was a yes/no question from Kathy to Bill about whether $r1$ was fresh.

5.3.2 Axioms used.

The first set of axioms for inheritance, contradiction detection and resolution, and the clock rule are likely to be common to a wide variety of problems described using this representation.

```
% inheritance rule: we inherit anything that is not killed and is not
%     itself a kill
(mt:(((‘(Q:X)) & (?(kill(X)))  & eval(\+ (Q = time)) & (?(k2(Q)))
      & eval(\+ (X = k2(_))) & eval(\+ (X = kill(_)))) => (Q:X)))@0.

%contradiction detection rule: when we detect a contradiction, we add
%     a contra belief at the next step and we kill both contradictands
%     and the contra belief itself (so these don’t propagate)

(mt:((‘(Q:X) & ‘(W:not(X)) & (?(kill(not(X))))) =>
    mt:(kill(not(X)))))@0.
(mt:((‘(Q:X) & ‘(W:not(X)) & (?(kill(contra((Q:X), (W:not(X))))))) =>
    mt:(kill(contra((Q:X), (W:not(X)))))))@0.
(mt:((‘(Q:X) & ‘(W:not(X)) & (?(kill(contra((Q:X), (W:not(X))))))) =>
    mt:(contra((Q:X), (W:not(X))))))@0.
(mt:((‘(Q:X) & ‘(W:not(X)) & (?(kill(X)))) => mt:(kill(X))))@0.

%contradiction resolution rule: we prefer utterances(beliefs)

(mt:((contra((imp:X), (bel:Y))) => (bel:Y)))@0.
(mt:((contra((bel:Y), (imp:X))) => (bel:Y)))@0.

% the clock rule
(mt: ((‘(time:now(T)) & eval(T1 is T+1)) => time:(now(T1))))@0.

% the initial time
(time: (now(0)))@0.
```

The next set involves specific rules for the implicature problems:

```
% usually when X informs us of P, we believe P.
(bel: (( (inform(X, Y, P)@_) & whoami(Y) & (?(ab2(X, Y, P)))
         ) => (bel:P)))@0.

% direct response to a yes-no question- we believe the response
(bel: (( (respond(X, Y, P, q_yes_no(P))@_) & whoami(Y) & (?(ab2(X, Y, P)))
           ) => (bel:P)))@0.
(bel: (( (respond(X, Y, not(P), q_yes_no(P))@_) & whoami(Y) &
         (?(ab2(X, Y, P)))  ) => (bel:P)))@0.

% indirect responses to yes-no questions- we try to figure what it means
(bel: (( (respond(_, X, P, q_yes_no(Q))@T) & whoami(X) & now(T) &
          eval(\+ P = Q) & eval(\+ P = not(Q)))
      => (qyn(Q):P) ))@0.

% if we have figured the answer, we make it an implicature and we stop
% trying to find out what the response meant. In this case, we just lose
% all the irrelevant beliefs we came across.
(bel: (( ‘((qyn(Q)):P) & (?(k2(qyn(Q)))) & eval(P = Q)) => imp:P))@0.
(bel: (( ‘((qyn(Q)):P) & (?(k2(qyn(Q)))) & eval(P = Q)) =>
    mt:k2(qyn(Q))  ))@0.
(bel: (( ‘((qyn(Q)):P) & (?(k2(qyn(Q)))) & eval(P = not(Q))) => imp:P))@0.
(bel: (( ‘((qyn(Q)):P) & (?(k2(qyn(Q)))) & eval(P = not(Q))) =>
    mt:k2(qyn(Q))  ))@0.
```

Finally we have rules that are specific for the examples under consideration:

```
% things in fridges are cold
(bel: ((‘(Z:infridge(X)) & (?(k2(Z))) & (?(not(cold(X))))) => Z:cold(X)))@0.

% things in fridges are small
```

```
(bel: ((‘(Z:infridge(X)) & (?(k2(Z))) & (?(not(small(X))))) => Z:small(X)))@0.

% things in fridges are dead
(bel: ((‘(Z:infridge(X)) & (?(k2(Z))) & (?(not(dead(X))))) => Z:dead(X)))@0.

% things in fridges are edible
(bel: ((‘(Z:infridge(X)) & (?(k2(Z))) & (?(not(edible(X))))) =>
       Z:edible(X)))@0.

% cold roses are fresh
(bel: ((roses(X) & ‘(Z:cold(X)) & (?(k2(Z)))  & (?(not(fresh(X)))))
       => Z:fresh(X)))@0.



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% initial beliefs

(bel: (whoami(kathy)))@0.
(bel: (roses(r1)))@0.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

The utterances considered are also represented in the same way. These are presented later.

5.3.3 Example 1

For example 1, we have these additional formulas:

```
% utterances

% kathy asks bill whether the roses are fresh or not at time 1
(utt: (q_yes_no(kathy, bill, fresh(r1))@1))@1.

% bill responds to kathy’s question by saying that the fridges are in the
% fridge (at time 2)
(utt: (respond(bill, kathy, infridge(r1), q_yes_no(fresh(r1)))@2))@2.

% then, at time 7, bill tells kathy that the roses are actually not fresh.
(utt: (inform(bill, kathy, not(fresh(r1)))@7))@7.
```

Some of the output trace for this case is presented below.

Our implementation in active logic produces the positing of an implicature followed by its cancellation as we move from (B) to (C). There are several steps between (B) and (C) and several more after (C). The output trace below shows that at step 9 (which occurs after the utterance (C)) a contradiction briefly appears. Then it is withdrawn. At this point Kathy has no belief at all whether or not the roses are fresh. Then the belief that they are not fresh is restored using a rule that expresses the maxim of Quality. Below we reproduce and comment in more detail on some of the output trace for this example.

Step 1

```
time:now(1)
bel:roses(r1)
bel:whoami(kathy)
utt: (q_yes_no(kathy,bill,fresh(r1))@1)
```

Kathy asks (A) **Are the roses fresh?**

Step 2

```
utt: (q_yes_no(kathy,bill,fresh(r1))@1)
bel:whoami(kathy)
bel:roses(r1)
time:now(2)
utt: (respond(bill,kathy,infridge(r1), q_yes_no(fresh(r1)))@2
```

Bill says (B) **They are in the fridge**.

Step 3

```
rel(fresh(r1)):infridge(r1)
bel:infridge(r1)
time:now(3)
utt: (respond(bill,kathy,infridge(r1), q_yes_no(fresh(r1)))@2)
bel:roses(r1)
bel:whoami(kathy)
utt: (q_yes_no(kathy,bill,fresh(r1))@1)
```

Kathy starts to think about what Bill's response means. The proposition **rel(fresh(r1)):infridge(r1)** means some thing like: With regard to the question whether or not **fresh(r1)** the fact that **infridge(r1)** could be relevant. The active logic system, in the course of its normal inheritance procedure will try to fire any rules that it can using **infridge(r1)**. This is our simplified model for Kathy's thinking about or trying to figure out the relevance of Sill's indirect answer.

Step 4

```
utt: (q_yes_no(kathy,bill,fresh(r1))@1)
bel:whoami(kathy)
bel:roses(r1)
utt: (respond(bill,kathy,infridge(r1), q_yes_no(fresh(r1)))@2)
rel(fresh(r1)):infridge(r1)
time:now(4)
bel:infridge(r1)
rel(fresh(r1)):cold(r1)
rel(fresh(r1)):small(r1)
rel(fresh(r1)):dead(r1)
rel(fresh(r1)):edible(r1)
```

Nothing interesting yet. Kathy is still thinking. She considers the relevance of the temperature, size, and edibility of the roses.

Step 5

```
rel(fresh(r1)):fresh(r1)
time:now(5)
rel(fresh(r1)):edible(r1)
rel(fresh(r1)):dead(r1)
rel(fresh(r1)):small(r1)
rel(fresh(r1)):cold(r1)
bel:infridge(r1)
rel(fresh(r1)):infridge(r1)
utt: (respond(bill,kathy,infridge(r1), q_yes_no(fresh(r1)))@2)
bel:roses(r1)
bel:whoami(kathy)
utt: (q_yes_no(kathy,bill,fresh(r1))@1)
```

Here at step 5 Kathy finds that the roses are fresh. That is the import of

```
rel(fresh(r1)):fresh(r1).
```

Inferring either this or

```
rel(fresh(r1)):not(fresh(r1))}
```

will stop Kathy's search for an answer to her yes–no question.

Step 7

```
time:now(7)
imp:fresh(r1)
bel:infridge(r1)
utt: (respond(bill,kathy,infridge(r1), q_yes_no(fresh(r1)))@2)
bel:roses(r1)
bel:whoami(kathy)
utt: (q_yes_no(kathy,bill,fresh(r1))@1)
utt: (respond(bill,kathy,not(fresh(r1)), q_yes_no(fresh(r1)))@7)
```

Now Bill says (C) **The roses are not fresh**.

Step 9

```
time:now(9)
mt:kill(fresh(r1))
mt:contra(imp:fresh(r1), bel:not(fresh(r1)))
```

```
mt:kill(contra(imp:fresh(r1), bel:not(fresh(r1))))
mt:kill(not(fresh(r1)))
bel:infridge(r1)
bel:not(fresh(r1))
imp:fresh(r1)
utt: (respond(bill,kathy,infridge(r1), q_yes_no(fresh(r1)))@2)
bel:roses(r1)
bel:whoami(kathy)
utt: (q_yes_no(kathy,bill,fresh(r1))@1)
utt: (respond(bill,kathy,not(fresh(r1)), q_yes_no(fresh(r1)))@7)
```

Kathy believes what Bill says, **bel:not(fresh(r1))** but now detects a contradiction. So neither the implicature **imp:fresh(r1)** nor this belief will inherit to the next step.

Step 10

```
utt: (respond(bill,kathy,not(fresh(r1)), q_yes_no(fresh(r1)))@7)
utt: (q_yes_no(kathy,bill,fresh(r1))@1)
bel:whoami(kathy)
bel:roses(r1)
utt: (respond(bill,kathy,infridge(r1), q_yes_no(fresh(r1)))@2)
time:now(10)
bel:not(fresh(r1))
bel:infridge(r1)
```

But the belief that the roses are not fresh is derivable from **utt:(inform(bill,kathy, not(fresh(r1)))@7)** at step 9 because Kathy believes everything Bill said. However, the implicature from (B), that the roses are fresh cannot be rederived; an utterance initiates a search for relevance and hence implicature only at the step it is perceived. In active logic we spread the reasoning over a sufficient number of steps to, so to speak, divide and conquer some of the complex thinking that happens during discourse understanding.

5.3.4 Example 2

The common background includes the defeasible belief that old roses are not fresh. We saw in the first example that the implicature at (B) was based on defeasible beliefs about things in fridges and about cold roses. This means that we could have a Nixon Diamond[161] after (D); there will be two (defeasible) implicatures: one that the roses are fresh and the other that they are not fresh. This is what happens in our trace as we pick up Kathy's thinking at step 7.

Step 7

```
time:now(7)
imp:fresh(r1)
bel:infridge(r1)
utt: (respond(bill,kathy,infridge(r1), q_yes_no(fresh(r1)))@2)
bel:roses(r1)
bel:whoami(kathy)
utt: (q_yes_no(kathy,bill,fresh(r1))@1)
utt: (respond(bill,kathy,old(r1), q_yes_no(fresh(r1)))@7)
```

Here is where Bill says (D) **They are old**.

Step 8

```
utt: (respond(bill,kathy,old(r1), q_yes_no(fresh(r1)))@7)
utt: (q_yes_no(kathy,bill,fresh(r1))@1)
bel:whoami(kathy)
bel:roses(r1)
utt: (respond(bill,kathy,infridge(r1), q_yes_no(fresh(r1)))@2)
imp:fresh(r1)
time:now(8)
bel:old(r1)
bel:infridge(r1)
rel(fresh(r1)):old(r1)
```

As before, Kathy begins to think about the relevance to her question of Bill's utterance. So she now thinks that the age of the roses is relevant to her question **rel(fresh(r1)):old(r1)**. She will begin searching for what that relevance could mean.

Step 9

```
rel(fresh(r1)):not(fresh(r1))
time:now(9)

rel(fresh(r1)):old(r1)
bel:infridge(r1)
bel:old(r1)
imp:fresh(r1)
utt: (respond(bill,kathy,infridge(r1), q_yes_no(fresh(r1)))@2)
bel:roses(r1)
bel:whoami(kathy)
utt: (q_yes_no(kathy,bill,fresh(r1))@1)
utt: (respond(bill,kathy,old(r1), q_yes_no(fresh(r1)))@7)
```

She happens to discover the relevance after only one step. It is that the roses are not fresh **rel(fresh(r1)):not(fresh(r1))**.

Step 10

```
utt: (respond(bill,kathy,old(r1), q_yes_no(fresh(r1)))@7)
utt: (q_yes_no(kathy,bill,fresh(r1))@1)
bel:whoami(kathy)
bel:roses(r1)
utt: (respond(bill,kathy,infridge(r1), q_yes_no(fresh(r1)))@2)
imp:fresh(r1)
rel(fresh(r1)):old(r1)
mt:kill(not(fresh(r1)))
mt:kill(contra(imp:fresh(r1), rel(fresh(r1)):not(fresh(r1))))
```

```
mt:contra(imp:fresh(r1), rel(fresh(r1)):not(fresh(r1)))
mt:kill(fresh(r1))
time:now(10)
bel:old(r1)
bel:infridge(r1)
mt:k2(rel(fresh(r1)))
rel(fresh(r1)):not(fresh(r1))
```

Here the contradiction is discovered between the the implicature from (B) and the more recent implicature from (D). So both are marked **kill** so they will not be inherited at the next step.

Step 12

```
utt: (respond(bill,kathy,old(r1), q_yes_no(fresh(r1)))@7)
utt: (q_yes_no(kathy,bill,fresh(r1))@1)
bel:whoami(kathy)
bel:roses(r1)
utt: (respond(bill,kathy,infridge(r1), q_yes_no(fresh(r1)))@2)
time:now(12)
bel:old(r1)
bel:infridge(r1)
```

Here Kathy has reached a state where she cannot infer anything helpful about her original question. It worked out this way because Bill said things which only implicated that the roses were fresh or not fresh. But both implicatures were cancelled. Active logic *can* allow one to infer again what one has just withdrawn. But in the model we have designed the search for the relevance or import of an utterance only begins at the step immediately following the perception of the utterance as relevant to a question. This captures the fact that it is utterances, not beliefs, that give rise to implicatures. It also captures the idea that cancellation just eliminates the implicature; any new implicature would have to wait for a new utterance. However, we are not prepared to say that this strategy is generally applicable.

5.4 Discussion of the solution

This problem, the solution and the representation of the solution differ from those in the presupposition problem case. However, the part of the solution relating to mistakes is similar.

In this example as well as the previous one, the detection of a mistake occurs when a direct contradiction is found. This implies that there is a mistake somewhere, but it is not clear what the mistake is. This logic, just like that one for the presuppositions, prevents the potentially mistaken formulas from being used to derive new formulas. The way it does it in this case though, is to disinherit it so that it does not appear in future steps.

The specific strategy to decide which belief is mistaken depends on the domain. In this implicature domain, the strategy is simply to prefer beliefs over implicatures. This is expressed as:

```
(mt:((contra((imp:X), (bel:Y))) => (bel:Y)))@0.
(mt:((contra((bel:Y), (imp:X))) => (bel:Y)))@0.
```

If there is a contradiction between an implicature X and a belief Y, then Y will be added at the next step. Since once a contradiction is noticed the formulas are disinherited, the implicature is not present in the next step.

At this point, we have discovered what the mistake is: it is the implicature. This is not explicitly asserted in the database in this simple approach to mistakes. It is sufficient, in this example, for the mistaken beliefs not to be present in the subsequent steps and that is accomplished automatically.

In the second example, we have a contradiction between two implicatures. In our simple approach, there is no way to resolve such a contradiction so both contradictands remain disinherited. In this case we have not identified what the mistake is: whether it is that the roses are fresh or that they are not fresh.

Once again, if there had been actions taken on the basis of the mistaken beliefs or if we had recursive mistakes, it would have been necessary to explicitly assert and reason about the mistaken beliefs. The example, as the previous one, shows, however, that in cases where the behavior is limited, one can get by with a much simpler way to handle mistakes.

5.5 Related work

5.5.1 Lascarides and Oberlander

[100] use a system of default rules to infer implicated temporal relations in discourses. One of their domain problems involves the conflict between rules that reflect the Gricean manner maxim and rules that reflect the Gricean relevance maxim. (They may or may not wish to characterize their rules in this way.) They employ what seem to be well motivated methods for prioritizing default rules in order to resolve conflicts. For the examples we have discussed our system also resolves some conflicts. Implicatures give way to literal assertions and conflicting implicatures lead to suspension of belief. It may be interesting that some of this behavior in our system emerges from a combination of other behaviors that embody Gricean maxims and principles of rationality. We have not yet investigated whether and how we would implement more explicit prioritization in active logic. There are, of course, important differences as to how cancellation works and how the effects of, say, prioritized default rules arise. As far as we can tell the methods under study by Lascarides, Oberlander, and Asher do not model the drawing and later cancellation of implicature as we do. Recalling our first example, the difference we emphasize between the dialog (A), (B), (C) and the dialog (A), (C), (B) is not important to theirs, as well as perhaps most other, nonmonotonic theories.

5.5.2 Green and Carberry

[70] consider task oriented dialogs. Their treatment of yes–no question dialogs influenced our relevance rules in Figure 2. They and others use abductive inference to constrain the search space when trying to derive a direct answer from the utterance of an indirect answer. We think we could have used abductive inference for the same purpose in our system. We chose instead to work with a perhaps less efficient strategy. Given a relevant proposition one simply looks for rules in which that proposition can be instantiated. Then one generates inferences from those rules until a direct answer is found (if it is). This method may have some cognitive validity. Perhaps some indirect answers are more difficult to appreciate than others and perhaps that is because people only spend a short time trying out inferences that combine what was said with what they know. We have nothing more to say beyond these speculations at this time.

As far as active logic goes, we could attempt to subsume any or all of the above methodologies. That is, we could attempt to implement their methods in active logic. This would be an interesting project.

5.6 Conclusion

This example although quite different in implementation from the presupposition work illustrates that the same approach to mistakes can be used in both cases. This adds evidence to the notion that there could be a general mistake handing algorithm that can be used on a wide range of problems. The specifics of the algorithm are to be provided by the domain under consideration, however, just as the behavior of the domain has to be specified. The behavior of an agent in case of mistakes can be seen as a more thorough representation of the domain than a representation that ignores mistakes.

Part III

Mistakes in agents

Chapter 6

Mistakes in agents–an introduction

In the previous part, we have seen the design and implementation of software systems that detect specific kinds of mistakes in language processing. In both examples, the software was focused on a narrow domain and had special data structures and procedures to deal with mistakes. The dialog manager application was implemented in a more general framework but the approach to mistakes was still, to some extent, specific to that application.

We now turn to agents that are not as domain specific as those, towards a design for a generic agent that can deal with mistakes appropriately in any domain given an adequate description of that domain. This description will include information as to how to react to mistakes.

The first step is to generalize the treatment of mistakes we have seen earlier. The only mistakes that concerned the logic in the previous applications were mistaken beliefs. Each of these applications had a different way of to handle these mistakes. We now want a general way to express the choice of which among a set of possibly mistaken beliefs is indeed mistaken. We also want a general way of repairing the KB once the mistake has been identified. These should be domain-independent and should be usable whatever domain we wish to model. Our solution is to use a non-monotonic language to specify defaults and preferences among the defaults. Domain information that is uncertain is to be expressed as defaults and the choice between such conflicting beliefs is to be expressed as preferences. The algorithms developed to apply defaults and handle the contradictions that signal mistakes depend only on the structure of the reasoning and not the domain.

These algorithms and representations were then implemented as an application of Alma. We discuss that implementation in some detail including the deviations between the algorithms and the implemented procedures in Alma.

The fact that the logic uses a non-monotonic language and has rules to apply defaults suggests that it can be used as some kind of non-monotonic logic. Ours is an implementation rather than a formally defined logic. Other characteristics that set our implementation apart from non-monotonic logics are that it is based on mistakes, that it is computable and that the answers it gives can change with more information and even with time.

We do not (yet) have a formal semantics for the system which makes it harder to compare with other non-monotonic logics. We have instead collected a set of examples of non-monotonic reasoning from the literature and organized that into a test-suite for non-monotonic reasoners. We tested out implementation against that test-suite and for those problems that were expressible and that could be computed in our formalism, intuitive answers were obtained. A related issue is whether intuition (even collective intuition) is good enough a measure of the goodness of non-monotonic logics. This led us to review some work on experiments on human default reasoning.

Finally, we sketch the design of a general agent that is meant to handle mistakes. Contrary to the cases considered earlier, we now have to deal with mistakes in intentions and actions as well as mistakes in beliefs. The propagation of mistakes from beliefs to intentions to actions and conversely becomes more prominent in this setting. We pay particular attention to the fact that the agent will need to act in the world. Having the agent logically decide each action it will take seems rather unrealistic given the complexity of logical reasoning. So we look towards a a hybrid system of a plan execution architecture coupled with a logic that will look over the execution and intervene in case of mistakes.

Chapter 7

Towards a time-situated non-monotonic reasoner

The above two problems dealt with essentially similar problems, though with different representations. In this chapter, we consider a more uniform approach to these sorts of problem. While there are particularities in the domain that need to be represented differently in each problem, there is a common approach to handling mistakes. In both cases, mistakes are detected by a contradiction in the KB and following that, a problem specific set of axioms is used to find the mistake and the mistaken formulas are then removed. We aim to find a general way to express the preference for the formula taken to be true and general procedures to do repair of the KB. We aim for these to be generally usable in case of mistaken beliefs. The formalism we choose to represent this sort of information is that of non-monotonic logics. We therefore develop an active logic based reasoner that uses the language of non-monotonic logics to deal with mistakes and beliefs in a general way.

We begin in the next section by considering in more detail the similarities and differences between the two above applications. From that information, we consider how the techniques used could be generalized and derive desirable properties of a system that would be a generalization of the above two systems. Next we review non-monotonic logics and examine how the language of non-monotonic logic can be used to fulfill the expressive requirements of our desired system. We then describe the language our system uses, followed by the rules of inference of the system. We then focus on how that system handles mistakes. Since the language of the system is a non-monotonic one, we next consider the extent to which our system could be used to do general non-monotonic reasoning. We end by reviewing some of the work in this area.

7.1 Similarities and differences

In this section we bring out in more detail the similarities and differences in the above two examples. Inference is done in a similar way and although the approach to mistakes is similar, the details are different.

7.1.1 Detection of mistakes

In both cases, mistakes are detected through something like a contradiction between formulas in the KB. These are not direct contradictions of the form ϕ and $\neg\phi$ but are determined by the representations chosen for each example. In the presupposition case, the contradictions are of the form `assert(p)` and `presup(not(p))`. These formulas are part of the context of the conversation, which is a list of formulas. In the implicature case, the formulas that are contradictory are of the form `bel:p` and `imp:(not(p))`. The formulas are now explicitly tagged to be part of contexts.

The essential circumstance that we want to detect is the same in both cases: the presence of ϕ and $\neg\phi$. However, the specific details of the problem and the representation chosen prevent us from using these direct forms in the KB.

7.1.2 Consequences of contradictions

In both cases, the detection of a contradiction indicates a possible mistake and the consequences of the possibly mistaken formulas are prevented from being used in further computations. In the presupposition case, once a contradiction has been derived, these formulas are labeled `SUSPECT` and since there are no rules for doing inference with `SUSPECT` formulas, there is no further inference from these. The solution to the implicature problem labels the possible mistakes with `kill`. This prevents the inheritance of these formulas from one step to the next which effectively prevents further inference from these formulas.

Both approaches accomplish the same thing. However, in the presupposition case, the formulas are still part of the beliefs of the agent, but in a suspect state. In the presupposition case, they are not part of the beliefs. Given that history is available, it is not clear that the beliefs have to be maintained in the current state as in the presupposition case, except perhaps for efficiency. On the other hand though, the system is likely to get less efficient when its KB grows large.

7.1.3 Resolving contradictions

Resolving the contradiction mainly involves determining which of the contradictands is mistaken. The approaches are similar in that there is a preference for a class of formulas over another. In the implicature case, formulas of the form `bel:p` are preferred to those of the form `imp:not(p)`. In the presupposition case, formulas `assert(p)` are preferred to formulas of the form `presup(not(p))`. This preference is expressed as axioms in the KB. In the implicature case they are:

```
(mt:((contra((imp:X), (bel:Y))) => (bel:Y)))@0.
(mt:((contra((bel:Y), (imp:X))) => (bel:Y)))@0.
```

And in the presupposition case they are represented as Prolog rules:

```
prefer(C1, C2, C1):-
C1::formula(assert(_)),
C2::formula(presup(_)), !.
prefer(C1, C2, C2):-
C2::formula(assert(_)),
C1::formula(presup(_)), !.
```

Here too, the representation chosen for each case influences the specification of the preference. These preferences are further restricted in that we can only arbitrate between formulas such that each comes from a rather large class of formulas. We cannot for instance handle the contradiction between two implicatures, for instance, `imp:p` and `imp:not(p)` in the implicature case.

7.1.4 Miscellaneous differences

The implicature approach explicitly inherits formulas at each step through an inference rule. The presupposition approach retains the state of the KB from step to step, implicitly inheriting all formulas. The data-structures used to represent the domain information is different and that affects the representation of the mistake handling procedures. These differences are not essential for the handling of mistakes but preclude a uniform approach to the problem to the problem.

7.1.5 Summary

In both cases, the method of handling mistakes is similar, but the details of the representation and therefore of the processing differ. Mistakes are detected through the presence in the LB of formulas of a specific kind. Once they are found, they are prevented from being used in further inferences. The mistake is identified based on a preference between classes of formulas asserted in the KB. These implementations are specific to the problem that they are meant to solve and don't seem to be easily generalizable.

7.2 Generalizing this behavior

We would like to generalize the procedure for dealing with mistaken beliefs. We want to have a uniform representation of the problems so that the agents can use the same algorithm to deal with mistakes of that kind for different domains.

The desirable behaviors are:

1. **Detection of contradictions** If each domain has its own representation of contradictions, we can't have a general way of detecting or handling mistakes. A solution is to make the representation the simplest possible so that instead of contradictions between `assert(p)` and `presup(not(p))` in lists of formulas or between `bel:p` and `imp:not(p)`, we detect contradictions between `q` and `not(q)`.

2. **Choosing mistaken formulas** The current method is to verify that the formulas are from different classes. In the presupposition case, we prefer `assert` to `presup` and in the implicature case, we prefer `bel:` to `imp:`. These are large rigid classes of formulas. This prevents expression of preferences between two different `assert` for instance, or if we want to sometimes prefer assertions and sometimes prefer presuppositions. The approach of expressing preferences between classes of formulas seems sound but the classes themselves need to have more flexibility.

3. **Expression of preferences** The preferences themselves are expressed in the KB as axioms that decide what to do in specific cases of contradictions. If there are many classes of formulas and possibly many combinations that give rise to contradictions, we will need many axioms expressing these preferences for each possible combination. We don't want to have to express all this information explicitly. The logic should be able to reason about the preferences to a certain extent.

4. **Defaults and uncertainty** Once could argue that all information, except perhaps for logic and mathematics, is uncertain[106, 67]. It is all subject to being mistaken and there should be no certain information in most of commonsense reasoning. Practically, however, a good deal of the information we have is typically taken not to be mistaken. We do believe that the sky is really blue although we are open to the possibility that that is not in fact the case. This is even more so for the rather limited agent that we are concerned with here.

   We should therefore be able to distinguish between the certain and uncertain information in each domain. In the above examples this is done implicitly through the preference formulas: implicatures and presuppositions are taken to be uncertain. We would like an explicit expression of this uncertainty.

Eventually, we would like an agent that can consider all of its knowledge as being uncertain and decide dynamically what is and what is not to be trusted at different times. The mistake mechanism should still be useful in this setting. For now though, we do something much simpler and explicitly flag uncertain formulas.

7.2.1 Properties of the system

To summarize the above discussion, the system we want needs to have the following properties:

- A uniform representation of contradictions.
- A uniform approach to resolve these contradictions.
- A way to state that some formulas are usually true and others are always true.
- A way to represent preferences between any number of classes of formulas of any size.

Other requirements of the system that stem from our aim of it being the on-board reasoner of an agent are:

- It should be executable, that is the language should directly specify the behavior of the language.
- It should accept new information as it executes
- It should reason in time.

Some advantages of having a uniform representation are:

- We can have a single mistake-handling algorithm that will be usable for a wide range of problems.
- Any properties we can show for that system will apply to all domains that use it.
- Once we get the representation and algorithms right, it becomes easier to encode other domains.
- We can add capabilities to the agent by simply adding axioms instead of redesigning the agent.

7.3 Using nmr to specify behavior in cases of mistake

The expressive requirements about are found to a certain extent in languages for non-monotonic reasoning[157, 108, 115, 129, 117, 18, 53]: a non-monotnonic reasoning language allows us to express that some formulas are to be taken as typically true and that some classes of formulas are preferred to others. However, non-monotonic languages do not typically represent facts about inconsistencies in the KB, neither do they consider KBs that grow in time or become inconsistent. A further problem with non-monotonic logics is that they are typically non-computable whereas we need to obtain answers fast enough to keep up with the world. We need to adapt the language of non-monotonic reasoning so that it fulfills our requirements. The reasoning system that results is different from typical non-monotonic logics.

7.3.1 Non-monotonic logics

Nonmonotonic logics are meant to capture the incompleteness and uncertainty of knowledge in a logic. In contrast to monotonic logics, the set of consequences of a theory in a nonmonotonic logic does not grow monotonically with the size of the theory. We might have $T \models \phi$, where T is a theory and $T \cup T_1 \not\models \phi$. This is meant to capture some aspects of commonsense reasoning where knowing that Tweety is a bird leads one to conclude that it flies but if one, in addition, learns that it is a penguin, this leads to a retraction of this conclusion. Details of some non-monotonic logics are provided below. In the following, we consider a generic non-monotonic language that has defaults and preferences between the defaults.

7.3.2 Useful features of non-monotonic languages

We would like to use the language of nonmonotonic logics (1) to make a distinction between certain and uncertain information in the domain axiomatization; (2) express information indicating a preference for beliefs in case we have contradictions; (3) in certain cases preventing the derivation of the mistaken beliefs in the first place. The language should provide a uniform way of doing this for a wide range of problems.

Certain and uncertain information

Non-monotonic logics give us a clear way to express certain and uncertain information. Certain information is represented as first order formulas whereas uncertain information is represented as defaults. The defaults need special inference rules for their application.

Preferences for resolving mistakes

In the examples above, the procedure to decide which of a pair of contradictands is mistaken was made in a more or less domain or problem dependent way. Non-monotonic logics can help in that they provide a standard language in which to express a preference for one formula to another. Two features that we will be using are the assertion that some formulas are defaults and that some defaults are to be preferred to others. If all the relevant domain information is encoded in this fashion, we can have a domain independent strategy to find the mistaken formulas. The domain dependent information as far as mistakes are concerned, is encoded in the same way for all domains.

If we have a mistake that involves a formula that is derived from a default and one that is derived from certain information, the formula that is derived from certain information is to be preferred and the mistake is the formula derived from the default. For instance, if the agent believes that Tweety does not fly because it is a penguin (and penguins don't fly by default) and also sees Tweety flying with its own eyes (which we consider certain), then it has to come to the conclusion that Tweety indeed does fly and that its default conclusion was mistaken.

If either of two formulas is mistaken and both are dependent on some (different) defaults, then if we know that one default is preferred to the other, the mistake is most likely to be with the less preferred default. The agent might believe that Tweety flies since it is a bird, but also that Tweety does not fly since it is a penguin. Both these beliefs are derived from defaults, but the default that penguins don't fly is to be preferred to the default that birds fly, so the mistake is that Tweety flies.

The language of non-monotonic logics allows us to express easily and generally these choices for resolving the mistakes. The language can be used in a a large variety of domains and a standard set of nonmonotonic logic handling procedures are needed.

Preventing mistakes

In addition to selecting which beliefs are mistaken, non-monotonic logics with their default and preference formulas can *prevent* mistakes from happening. If we know for a fact that Tweety does fly, then even if we know it is a penguin and that the "penguins usually fly" default is applicable, we can refrain from applying this default since we know that it is going to be mistaken. Similarly, if a more preferred default has already been applied, we need not apply a less preferred one that will be removed later. The explicit representations of defaults and preferences is essential for that.

7.3.3 Problems with non-monotonic logics

Although non-monotonic languages enable us to express information that is useful in resolving the mistakes, there are problems with the typical non-monotonic logic that makes them unusable for our purposes. These relate mainly to the on-board agent logic character that we aim for and that is essential for practical applications.

No time-situatedness

Traditional non-monotonic logics are not time-situated. They can reason about time and in fact do so in the case of temporal reasoning, but do not reason in time. An on-board logic for an agent needs to reason in time though. It has to take the passage of time into account in its computations. One important aspect of this is that it needs to make decisions and act before it is too late. This may mean that there is not enough time to compute the best answers to queries.

Incorporating new information

Nonmonotonic logics are typically provided with a static description of the world for them to reason about. In our case, we have a dynamic set of facts about the world that is fed into the logic. There is never all the information that will be relevant and new information in constantly coming in that can make previous reasoning invalid. This could be done in non-monotonic logics by constantly recomputing the logic with new information but that would then be ineffective. The logic needs to be able to incorporate new and potentially inconsistent information while it is running and do so efficiently somewhat like belief revision systems [69]

Non-computability of nmr

A feature of non-monotonic logics is that they are generally uncomputable [22, 24]. They are therefore not very useful for agents that are acting in the world and need to decide *now* whether Tweety flies or not and are prepared to make mistakes.

One way to see this is a problem is that we will have to make decisions and come to conclusions before we have enough information to do so. Knowing only that Tweety is a bird, we will conclude, if necessary, that Tweety flies even though it could be a penguin and therefore not fly and we don’t know that that is not the case. This simply means that the agent will be liable to commit mistakes in its reasoning. That is not a problem for us since we are set up to deal with mistakes. Non-monotonic logics though, are not usually designed for these circumstances.

Meta-reasoning

The mistakes made by the reasoner can be reasoned about if we have a theory about mistakes. To do that in the same logic that reasons about the domain of interest requires the capability for meta-reasoning. This is not typically available in non-monotonic logics.

The advice we can obtain from the NMR as to which mistake to prefer can itself be seen as potentially mistaken information. If it does turn out to be mistaken, we can deal with this mistake just as we deal with any other mistake.

Seeing the NMR as an incomplete and uncertain specification of what to do when there is a mistake in the KB unifies this view.

Contradictions

A consequence of the need to make decisions before all relevant information is known and the non-computability of the logic is that there are bound to be contradictions and inconsistencies in the KB. Non-monotonic logics are typically based on complete first order logics and therefore the presence of an inconsistency makes all the formulas in the language derivable. Further the contradictions or inconsistencies cannot be reasoned about or resolved. This makes such logics unusable for our purposes where mistakes and inconsistencies are expected.

7.4 Language

Since the requirements of our system differ significantly from non-monotonic logics, we cannot use these. We will however use the language of non-monotonic logics to specify the domains we want to model. We now describe the non-monotonic language of the logic that we build. The logic is non-monotonic, but it also has various other properties that are essential to it being used as an on-board logic.

Our language is like a first order language with a new connective representing defaults ($\hookrightarrow$), a naming operator that relates formulas to names, a preference relationship that applies to names of formulas and is intended for defaults.

7.4.1 The default operator

If ϕ and ψ are formulas, then $\phi \hookrightarrow \psi$ is a formula. We intuitively understand that as representing that if we believe ϕ to be true and there are no facts that would suggest ψ not to be true, then it is reasonable to conclude that ψ is true. If we know of other facts that suggest that $\neg\psi$ might be true, then even if ϕ is true, we may not want to say that ψ is true. The other relevant facts can be other defaults or first order formulas.

For example, if we have $Bird(Tweety) \hookrightarrow Flies(Tweety)$ and we know that $Bird(Tweety)$, we can conclude $Flies(Tweety)$. If, however, we know that $\neg Flies(Tweety)$, then we do not want to conclude $Flies(Tweety)$. In this case we have a fact, $\neg Flies(Tweety)$, that suggests that $Flies(Tweety)$ may not be true. Or if we have $Penguin(Tweety)$ and $Penguin(Tweety) \hookrightarrow \neg Flies(Tweety)$, that too suggests that Tweety might not fly and therefore we do not want to apply the default without further consideration (see later).

These formulas can be universally quantified. We then get, for example, $\forall x\ Bird(x) \hookrightarrow Flies(x)$. This simply generalizes the above for each object that we know of. So, given any object, if we know it is a bird and we do not know anything that would suggest that it might not fly and we can conclude that it flies.

Universally quantified defaults are the way we would typically express regularities in the world: birds fly, penguins don't fly, summer days are warm and so on. These can then be instantiated to give *default instances*. Given $SummerDay(x) \hookrightarrow Warm(x)$, we can instantiate that with $SummerSolstice$ so that the default instance is $SummerDay(SummerSolstice) \hookrightarrow Warm(SummerSolstice)$.

We will generally refer to a default $\alpha_i \hookrightarrow \beta_i$ by δ_i. α_i is the premise of the default and β_i is the consequent of the default. The instantiation of the default with c is referred to as $\delta_i(c)$. (We sometimes omit the "(c)" and refer to this as δ_i.)

Applying a default

The logic can apply a default instance $\delta_i(c)$ if $\alpha_i(c)$ is in the KB. The result of applying the default is the addition of $\beta_i(c)$ to the KB. If $\delta_i(c)$ is a universally quantified default, we can talk about applying δ_i to some object a which is understood as instantiating δ_i with c and applying the resulting instantiated default. For instance, we can apply the instance $SummerDay(SummerSolstice) \hookrightarrow Warm(SummerSolstice)$ if $SummerDay(SummerSolstice)$ is true with the result that $WarmDay(SummerSolstice)$ will be added to the KB.

A default holding

We say that a default $\delta_i(c)$ holds if $\alpha_i(c)$ and $\beta_i(c)$, both are in the KB.

7.4.2 Names

Formulas can be assigned names which makes it easier to refer to them later. Referring to formulas is essential for meta-reasoning and is not usually available in non-monotonic logics. An operator $Named$ is used for that. The name of the formula can be parameterized. This allows instances of formulas to have instantiated names. Some examples:

- $Named(Bird(Tweety) \hookrightarrow Flies(Tweety), TweetyFlies)$ The name of the formula $Bird(Tweety) \hookrightarrow Flies(Tweety)$ is $TweetyFlies$
- $\forall x\ Named(Bird(x) \hookrightarrow Flies(x), BirdFlies(x))$ In this case, each instance of $Bird(x) \hookrightarrow Flies(x)$ has a name $BirdFlies(x)$ The default used to conclude that Tweety flies, for instance will be $BirdFlies(Tweety)$.
- $Named(\forall x\ Bird(x) \hookrightarrow Flies(x), AllBirdsFly)$ In this case, the name of instances of the formula are not specified. This only expresses the name of the quantified default. The instantiated defaults cannot be referred to by the name specific to the instance in the description of the domain. Instantiation of formulas with parameterized names results in the names too being instantiated.

7.4.3 Preferences

Preferences between defaults can be expressed using the $Prefer$ predicate. $Prefer(N1, N2)$ says that formula (default instance) $N1$ is preferred to $N2$. The preference is useful to express preferences between defaults that may be conflicting. If we have two defaults that are relevant to some formula β_i, then the preference can tell us which to apply.

For example, we can have a default instance that if Tweety's birthday is a summer day, then that day will typically be a warm day:

$$Named(SummerDay(TweetyBirthday) \hookrightarrow WarmDay(TweetyBirthday), WD1(TweetyBirthday))$$

On the other hand, if there is a cold front that day, it will not be warm:

$$Named(ColdFrontDay(TweetyBirthday) \hookrightarrow \neg WarmDay(TweetyBirthday), WD2(TweetyBirthday))$$

If we know that Tweety's birthday is a summer day but that there is also a cold front on that day, we can apply neither of these defaults because there is some other fact that we know that is relevant to the warmth of the day. But if we also know that

$$Prefer(WD2(TweetyBirthday), WD1(TweetyBirthday))$$

We should be able to use that to conclude $\neg WarmDay(TweetyBirthday)$.

Preferences can be quantified, so that $\forall x\ Prefer(WD2(x), WD1(x))$ can be instantiated with $TweetyBirthday$ to give the instance of the preference above.

Preferences, like defaults, are assumed to be provided as part of the domain description.

7.4.4 Inconsistent Default Sets

It is possible to have a set of defaults such that not all of then can hold in the KB at the same time and there is no preference between them, for instance the Nixon Diamond [161]. The defaults cannot all hold because that would cause an inconsistency. Because not all the defaults can hold and we have no evidence as to which ones should or should not hold, we have no opinion about any of the defaults holding. Such a minimal set of defaults that are jointly inconsistent and between which there are no preferences is called an ID (inconsistent default set). See below for an example.

7.4.5 Defeasible and non-defeasible parts of the KB

The first order formulas and their consequences form the non-defeasible part of the KB. Once a non-defeasible formula is derived, it is never removed from the KB. Defeasible formulas–those derived from defaults–can however be removed from the KB in case of mistakes.

7.4.6 Usage

The above features make many new formulas possible in the language. The intended use is illustrated here.

We have a number of named defaults:

$$\forall x\ Named(Penguin(x) \hookrightarrow \neg Flies(x), PenguinsDontFly(x))$$

$$\forall x\ Named(Bird(x) \hookrightarrow Flies(x), BirdsFly(x))$$

The preference between these can be expressed as:

$$\forall x\ Prefer(PenguinsDontFly(x), BirdsFly(x))$$

If Tweety is a penguin, $Penguin(Tweety)$, we will instantiate both defaults with resulting names $PenguinsDontFly(Tweety)$ and $BirdsFly(Tweety)$ and the preference instantiated with Tweety is $Prefer(PenguinsDontFly(Tweety), BirdsFly(Tweety))$. This allows the logic to choose between the contradictory consequences.

Another example:

$$\forall x\ Named(Quaker(x) \hookrightarrow Pacifist(x), QuakerPacifist(x))$$

$$\forall x\ Named(Republican(x) \hookrightarrow \neg Pacifist(x), RepublicanNotPacifist(x))$$

This time we do not have a preference between the defaults. If we know that $Quaker(Nixon) \wedge Republican(Nixon)$, we don't want to conclude that Nixon is or is not a pacifist. The result is that $\{QuakerPacifist(Nixon), RepublicanNotPacifist(Ni$ is an ID. Note that it is the instances that are in the ID and not the general defaults.

7.5 Rules of inference

We want our inference rules to only allow the derivation of formulas that are to be believed according to the intuitions presented above. This typically means not applying all the defaults with true antecedents because although a default may be applicable, there might be a suggestion that the consequent β is not true. As seen earlier, these might be that 1. $\neg\beta$ is derivable non-defeasibly or 2. there is a more preferred default with antecedent that is true and whose consequent implies $\neg\beta$ or 3. there is a set of defaults with inconsistent consequents and the default of interest is part of that set. To be sure that we can apply a default, the logic needs to verify these conditions beforehand. These conditions involve proving that some formula is not provable: in case 1 that $\neg\beta$ is not non-defeasibly provable; in case 2 that there is no preferred default δ_j such that $\alpha_j \wedge \beta_j$ is provable; in case 3 that there is no set of defaults such that they hold and $\neg\beta$ is provable from them. So these conditions cannot be computed in general. There are two choices: apply no default or apply a default and detect and repair the possible mistakes. We choose the second approach and therefore we cannot achieve the aim to only apply defaults that should be applied since we will occasionally apply a default that should not be applied. We instead now aim to approach the state of having only the defaults that should be applied applied with enough computation.

7.5.1 Approach

The approach taken is to verify that the required conditions hold *in the current state of the KB*. This can of course introduce mistakes since a condition that might seem to be true given the current state of the KB, may not be true after all because not enough computation had been done at the point the condition was tested. For instance, we might know that Tweety is a bird and therefore can conclude that it flies, but we might also know that it was born in 1900 but have yet to conclude that this means Tweety cannot fly. We only consider mistaken beliefs so the mistakes manifest themselves as inconsistencies in the KB. Inconsistencies are detected as direct contradictions–between ϕ and $\neg\phi$. The occurrence of a direct contradiction indicates that at least one default has been mistakenly applied, but not which one. We then have to identify the mistake and repair the KB.

In each of the cases that we apply a default that should not be applied, a contradiction will be derived. Say we apply δ_i. If that is a mistake because $\neg\beta_i$ is non-defeasibly provable, we will eventually prove this and there will be a direct contradiction. Similarly for the cases of a more preferred default holding and δ_i being in an ID. Since we assume that the non-defeasible parts of the domain axiomatization are consistent, defaults and the misapplication of defaults is the only way contradictions can be obtained. So, we obtain contradictions if and only if we misapply defaults.

The rules that we need to specify then are a default application (DA) rule and a contradiction resolution (CR) rule. Note that the derivation of a contradiction is a very useful and important event since this is what indicates to the logic that it has misapplied some default and is therefore mistaken about some of its beliefs.

7.5.2 AL rules

The default application rule has to verify that the conditions mentioned above hold. Detection of the direct contradictions is done by the active logic engine which asserts that there has been a contradiction as `contra(n1, n2, t)`, where `n1` and `n2` are the names of the contradictands and t is the time at which the contradiction was detected. Active logic also distrusts the formulas and their consequences which would not have been derived were it not for the contradictands. Further computation, including identifying the mistaken beliefs and restoring the true formulas is specified by the CR rule.

DA: Applying a default

The rule for applying a default is straightforward. Given a default (instance) $\delta_i = \alpha_i \hookrightarrow \beta_i$, if α_i is in the KB, we add β_i to the KB unless either of these conditions hold:

- $\neg\beta_i$ is in the KB and has been derived non-defeasibly. This can be computed by verifying that $\neg\beta$ is in the KB and examining its derivation for the presence of any defaults.
- There is some default δ_j such that $Prefer(\delta_j, \delta_i)$ and δ_j holds in the KB. This is easy to verify too since the logic represents the preference relations explicitly and it is easy to verify whether α_j and β_j are in the KB.
- There is some ID that that default is in. This can be done simply by looking up the IDs recorded by the logic.

With this rule, defaults are not added if it is apparent that there is a stronger reason not to do so. The application of a default is clearly computable. it mainly involves KB lookups. The cost for this fast default application or jumping to conclusions is that we may be mistaken. We do not have the assurance that once a default is applied it will hold forever. But this is a cost we are willing to pay since we can handle the mistakes and the alternative–not to apply the default unless we are sure it is correct to do so–condemns us to not doing anything.

However, when a default is applied, we also need to verify whether there are less preferred defaults that holds in the KB and that should not. Taking this approach rather than allowing the contradiction to appear later simplifies the CR rule and reduces inconsistency in the KB. So another task for the rule is:

- Given that δ_i has been applied, for each default instance δ_x such that $Prefer(\delta_i, \delta_x)$, if δ_x holds in the KB, it is removed and its consequences undone. See later for details of undoing.

The parameterization of the names of defaults with variables in the default itself becomes useful here. Consider the following example:
$\forall x\ Named(Bird(x) \hookrightarrow Flies(x), BirdsFly(x))$ and
$\forall x\ Named(Penguin(x) \hookrightarrow \neg Flies(x), PenguinsDontFly(x))$ together with the preference
$\forall x\ Prefer(PenguinsDontFly(x), BirdsFly(x))$.
If we apply of $PenguinsDontFly$ to Joe, we do not want that to prevent us from applying $BirdsFly$ to Fred because of the preference. This is not going to happen here because the defaults applied are $PenguinsDontFly(Joe)$ and $BirdsFly(Fred)$ which cannot be the arguments of any instance of the preference.

CR: Resolving contradictions

The other rule that we need is the CR rule to respond to contradictions. These indicate mistakes in reasoning, or more precisely, mistakes in applying defaults. Once the contradictions are detected, the contradictands and their consequences are distrusted automatically by Alma which results in these formulas not generating any new consequences. The CR rule needs to find the cause of the mistake and repair it. The repair involves reinstating some formulas and deleting others. It also serves to detect IDs.

The Alma automatic response reflects a realization that there is a mistake and that the mistaken formula is either of the contradictands, but the active logic engine is in no position to decide which it is. The first task of the CR rule is to find out the which of the contradictands is mistaken. This is a diagnostic task that is compiled into the inference rule. This is possible since there are just a few ways that this could happen.

From the default application rule, we can see that a clash among defaults that have preferences among them will not result in a contradiction because the less preferred default is removed when the other is added. Therefore the only possibility is an ID: a set of defaults is jointly inconsistent and such that there is no preference among these defaults.

The problem then is to decide from all the defaults involved in the inconsistency, which are those that are to be taken to be in the ID. We need to find the minimal set of such defaults. Note that the ID could be a singleton set in which case this is a default the negation of whose consequent is obtained non-defeasibly from the KB.

Computing IDs The defaults involved in the inconsistency are simply the defaults appearing in the derivation of that inconsistency. If there are multiple derivations of some formulas, we might get several sets of defaults. Defaults not appearing in the derivation are not related to the contradiction and are of no concern.

The sets obtained need not be minimal sets. Assume we have a contradiction between ϕ and $\neg\phi$ and the defaults in the derivation of ϕ are δ_0 and δ_1 and for $\neg\phi$, there is just one default, δ_2, involved. If δ_1 depends on δ_0 for it to hold, that is δ_0 is in the derivation for β_1, then δ_0 cannot be in the ID since once we assume δ_1 holds, the inconsistency is present. The ID consists of δ_1 and δ_2. In general then, we identify the IDs as the defaults that are the closest to the contradictands–the "leaf defaults". This can easily be computed from the derivation tree. The IDs are then asserted in the KB.

There is no problem of getting too small a set of defaults in the ID since the leaf-defaults include the first ID on every branch of the derivation for the contradiction. Those defaults not included are used to derive the conditions that allow the defaults in the ID to hold. These defaults, by themselves, are not directly involved in the contradiction and should not be included in the ID. As long as the premises for the defaults in the ID hold, whether this is though some default or not, we will get the inconsistency. Other defaults not on the derivation tree are not involved in the contradiction either. So the IDs computed are not too small.

The computation of the ID can also be mistaken. It may be non-minimal because a default included might not be relevant to the contradictand. Assume β_0 is a contradictand which is obtained from $\beta_0 \wedge \beta_1$ which in turn was obtained from β_0 and β_1. According to the procedure described above, both δ_0 and δ_1 will be in the ID, but δ_1 is irrelevant to the contradiction. Since we only use β_1 to get $\beta_0 \wedge \beta_1$, from which we then re-derive β_0.

The solution is to try to prove that subsets of the consequents of the IDs we computed are inconsistent using only non-defeasible formulas. If any of these proofs succeeds, the subset is an ID and the previously computed one was mistaken. The new ID could also be mistaken though and similar proofs are started. If none of the proofs ever succeed, we have a minimal ID. However, the logic can never be sure of that.

These proofs are done in parallel with the usual computation of the logic and we do not wait for them to succeed or fail before going on with using the ID.

The CR rule
Therefore, when a contradiction is found, the CR rule does the following:

1. Find the leaf defaults.
2. If there is just one, undo the effects of detecting the contradiction and undo the application of that default. In this case, there is just one default and the negation of its consequence is derived non-defeasibly. This condition is verified by the DA rule before applying a default and this default instance will therefore not be subsequently applied.
3. If there are several leaf defaults, undo the effects of the contradiction detection rule and undo the application of the leaf defaults. Further, record the set of distrusted formulas in an ID and start proofs to look for subset IDs. If any of these succeeds, undo the distrust of the defaults that are in the ID and take this to be a new ID.

Undoing consequences

In the above, there is the need to undo the consequences of some action at several points which is where the consequences of the mistake get repaired. The consequences are the actions taken in the KB–addition or deletion or distrusting of formulas. We assume that these actions are recorded in the Alma history and that they are accessible given the formula. The aim of the repairs in this domain is to remove formulas now believed to be false from the KB and to add the true formulas that were removed (or ensure that they can be later added) because of the mistake. These operations should be done recursively on the consequences of the formulas added or deleted. There are some complications however which we address now.

Not deleting everything added In undoing the addition of formulas, the general rule is that if some formula depends on a formula now believed to be false, then that formula too is false. This is correct generally, except for special cases as the `contra` formulas or other historical formulas. The `contra` formula expresses that there *has been* a contradiction at time `t`. Even if we do not believe the formulas anymore, the fact that there was a contradiction still holds and we should not delete that `Contra` assertion. One would in general do this by reasoning about each of the formulas to decide whether it is false or stays true.

In the case of this logic though, since there are a small fixed set of such cases, we can compile this reasoning into the procedures for undoing consequences so that the `contra` formulas stop the recursive undoing. This reduces the flexibility of the mistake processing but increases efficiency. The reduction of flexibility is not important if we reason in the same system, but if we want to change the way the reasoning is done, that may be important.

Undoing undoes Since the belief that there is a mistake might itself be mistaken, we might need to undo some previous undoes. This does not present special problems except that all the actions done as a result of the previous undo should be recorded and those actions undone.

Multiple derivations If the default application to be undone has a consequence ϕ but ϕ is also independently derived, then ϕ and its consequences should not be deleted. We can think of adding or deleting derivations rather than formulas. Then, it there are no more derivations for a formula, it is deleted.

Removing IDs When there is a set of defaults that are jointly inconsistent, one of the actions done is to assert that there is an ID. However, if one of the defaults in the ID is found to have been applied by mistake, for example if the negation of the consequence is derivable non-defeasibly, then the assertion that that set of defaults is an ID is not true anymore. We need to ensure that that assertion is removed as a result of the removal of any of the defaults in it even though the addition of the ID assertion was not directly triggered by the addition of that default.

7.5.3 Characterization

We now summarize these rules by stating the circumstances under which there is a derivation of a default consequence, or the deletion of the consequence and the derivation of IDs. We assume for simplicity that there is a unique derivation for the formulas under consideration.

β_i is added to the KB at time t if all of these conditions hold

- There is no non-defeasible derivation of $\neg\beta_i$ at time t and
- There is no derivation of $\neg\beta_i$ that depends on a more preferred default at time t.
- α_i is in the KB at time t
- δ_i is not in any ID at time t

If β_i is in the KB at $t-1$, it is removed at time t if either

- $\neg\beta_i$ is derived non-defeasibly at time t or
- $\neg\beta_i$ is derived from a more preferred default at time t or

- δ_i is found to be in an ID at time t

$\delta_i, \ldots, \delta_j$ are asserted as being members of an ID if there is a derivation of a contradiction that involves these as leaf defaults.

$\delta_i, \ldots, \delta_j$ are removed from the list of IDs if for any one of the defaults, either the antecedent is removed from the KB or if one of the defaults preferred to this one is applied to the KB or if a subset of this ID is inconsistent.

The logic will not know when it has the "final answer" to any query unless this can be non-defeasibly derived. If that is not the case, there is always the possibility that a new derivation will arise that will make the previous answer invalid. So the logic can typically not "know" that it has the right answer even if it has it, and the answer can change over time. However, a later answer, incorporates the information used to obtain all the earlier ones. This implies that later answers are better than earlier ones so that our confidence in the answers should increase too.

If the domain is one in which there are rather short derivations that will be of interest to the problems in the domain, then assuming a fair control strategy for choosing which inferences to do, we will eventually compute all the relevant derivations and we can expect the answer to stabilize after some time. The logic still cannot claim to know that this is the correct answer, but that will typically be the case.

7.6 Mistake handling in this logic

While our logic with a non-monotonic language admits of a great variety of domains, the structure of these domains is more circumscribed than the ones that we can represent in active logic in general. This allows more precise methods for dealing with mistakes.

Some of these mistakes are simple enough that we can deal with them immediately and any mistaken responses can be handled in the same way. In this case, these mistakes are not explicitly represented or reasoned about logically and the logic has special procedures in the inference rules to generate the correct response. This illustrates that not all mistakes all the time should be explicitly represented and reasoned about in the logic. It is more efficient not to do so when we can, but some flexibility is lost. In this case, this is not a problem.

7.6.1 Detecting mistakes

As in the case of active logic, mistakes are detected by contradictions. The assertion of the `contra` formula represents that one of the contradictands is a mistaken belief, and so does the assertion of the `distrust` formulas. In this case however, since the structure of the representation is determined by the logic, it is possible in some instances to pin down which formula is mistaken. This is helped by the assumption in the logic that the only source of mistakes are the defaults.

If one of the contradictands involves no defaults but the other one does, it is clear that the mistake lies in the default application that gave rise to the contradictand. That default, together with all its consequences are mistaken.

In the case that there are more than one default involved in the contradiction and no preference information is available, the logic falls back to a general strategy by distrusting all the (leaf) defaults involved and their consequences. This is still more informed than the case in active logic because we can here move up the derivation tree to find the possible causes of the mistakes.

An important case of detecting mistakes is the implicit detection that there has been or will be a mistake in applying defaults. In the first case, the application of a more preferred default involves the search for a less preferred one that has been applied earlier. This amounts to detecting a mistake before it derives a contradiction. In the second case, the attempt to apply a default involves a search for a more preferred default that may have been applied. This too prevents a future mistake that would be obtained by applying the less preferred default. These two cases of mistake detection do not appear explicitly in the KB and immediately result in the appropriate mistake handling procedure.

Another sort of mistake that is detected is that of taking a superset of an ID to be an ID. The detection occurs when a proof that a subset of that set is inconsistent succeeds. This is not directly represented in the KB and is handled immediately. This is another instance of a small number of fixed mistakes that can be handled procedurally. The domain of these mistakes this time is not the domain axiomatized but the logic itself. If the new ID that results from this mistake detection is itself mistaken, the same procedure can be followed and we do not need to undo any past responses.

7.6.2 Handling mistakes

Handling mistakes too is done in a more informed way here as compared to the case for active logic. Corresponding to each of the methods for the detection of mistake described above, there is a mistake handling behavior.

In the case of implicit detection of mistakes, the logic immediately deletes the consequence of the weaker default application and removes all of its consequences from the KB. Another preemptive move is for the logic to refrain from applying a default that is less preferred than one already applied.

If a contradiction is detected, this means that there has been no preferences between the defaults. If there is just one default involved, its application has to be mistaken and the problem is solved. Otherwise, the logic cannot decide between the defaults involved and all the leaf defaults are distrusted. The lack of information leads to the logic having to represent the possibility of a mistake rather than taking action to solve the problem.

Handling the case of mistaken IDs is done automatically–the reasoning in this case is simple enough that there is no need for logical inference and a procedure is sufficient.

7.6.3 Representation

In the cases that the logic can definitely identify the mistake, it is dealt with without any representation of it being made in the KB. In the ID case however, we need to keep the representation of the mistake since the logic does not know what is mistaken and the ID itself could be mistaken.

The lack of representation for the case that the logic can detect and resolve the mistake implies that the logic cannot later reason about this decision in case there are mistakes in the decisions of what is mistaken. This is not a problem in this logic but in a more general domain where one may need to reason about the preferences, possibly using defaults, this approach will limit the ability of the system to repair some of its mistakes. In that case, because the preferences might be mistaken, finding a mistake in a preference might mean undoing the consequences of that preference. To do this, a record of the consequence of the preference will have to be kept which is not done in the rules presented above. The ongoing Alma history record does have a record of all derivations of interest, however these are not as convenient to work with as more explicit records.

The mistaken IDs is an example of knowing what the mistake is and how to handle it so that there is no need to assert it in the KB.

7.7 Using this as a general NMR

The aim of this chapter was to use the language of non-monotonic logics to specify the behavior of the system in the event of mistakes. We now consider how the resulting system can be used as a non-monotonic reasoner in its own right.

7.7.1 Differences

There are several differences between the system we have described here and a non-monotonic reasoner:

- Our reasoner can give an answer relatively fast.
- However any answer can be mistaken.
- It can accept new information as it computes
- It tolerates inconsistencies
- It is a mainly forward chaning reasoner

We can in general not tell when the system will have derived the final answer and whether the answer it derives is the correct one. This is caused by the UN-computability of non-monotonic logics in general and on our dependence on non-provability formulas. If we could be sure in general when the system comes up with the right answer, the process would be computable, which may be the case for restricted fragments of non-monotonic logics, but not for general logics as we consider them here. This is the case even if no new information is added to the KB. The answer can change just because more computation is done and new facts emerge. So the system we have will have changing answers and there is no way to know when it has the correct answer. This does not look promising as a system to compute non-monotonic logics.

However, the later answers the system derives are based on all the information the earlier ones were derived plus new information that made the logic change its KB. The later answers are then based on more information than the earlier one and can be seen as being better answers (provided the old and new information are combined appropriately). The system then seems to act as an anytime [16] non-monotonic reasoner–the quality of the answers improves with time.

If we further assume that the derivations that are relevant to the computations we are interested in are rather short, given an appropriate control strategy for applying inference rules, we can expect the answers not to change anymore after a relatively short time. Those answers are likely to be the correct ones. There is evidence that the sort of reasoning done in everyday situations tend to be short [140]. It seems that these “everyday” situations are commonsense ones. If the control strategy extends derivations uniformly, it satisfies the second condition. We can then expect to have the correct answer for commonsense problems in a relatively short time. Further work has to be done to formalize these notions.

If we accept the fact that some answers can be mistaken and that we should prefer later answers to earlier ones, then given the appropriate domains–those where the relevant derivations are short, this can be used as a non-monotonic reasoner which gives good answers relatively fast.

7.7.2 Tests

We have tested the reasoner on a large variety of examples of non-monotonic reasoners in the literature. The results of the tests are discussed later. In summary, if we are willing to wait for the correct answer to be derived, this system does generally behave as a non-monotonic reasoner intuitively should.

7.8 Related work

In the 1980's a number of new logics were devised for nonmonotonic reasoning. We give a brief account of a few of them here. All of the approaches we describe except for the more recent work were more concerned with defining a logic that could formalize the intuitive jumping to a conclusion that seems to be needed to reason in domains other than mathematics and logic. These approaches were, in general, not computable, and some were not even meant to be computable despite the advertised aim to model jumping to conclusions. More recent work has focused on a computable approaches to the problem. However, these do not quite jump to conclusions the way one would expect. This makes the logics not quite suitable for use as an on-board logic for agents.

The approach we have presented here, although it has problems with controlling the inference is much closer to being the sort of logic that can be used to control an agent acting in the world. The main problem with that approach is that we do not have a guarantee that we have the right answer to a query because the answer we have is liable to change later with more information or more computation. This sort of behavior seems to occur in people too: we might decide to make coffee but with more thought we might decide to drink tea instead. If people, who after all, are the paradigms of commonsense reasoning can change their minds, it is surely acceptable for agents to do so too. The other problem with our approach at this time is that we do not yet have a formal semantic characterization of the logic. This is an important aspect of the logic but one that may not be as crucial for our approach as problems of control of reasoning are.

7.8.1 Circumscription

McCarthy [110] uses circumscription to formalize the process of jumping to conclusions. Predicate circumscription is a rule of conjecture that allows one to jump to the conclusion that the only objects satisfying a property P are those that can be shown to satisfy P. Circumscription is nonmonotonic since by adding a new object a that satisfies P to the knowledge base, we can no longer conclude by circumscription that a does not satisfy P even though we could do so before the addition.

The circumscription of P in $A(P)$ (where A is a first order sentence containing predicate P and $A(\Phi)$ is obtained by replacing all occurrences of P in A by Φ), is given by the sentence schema:

$$A(\Phi) \land \forall\overline{x}(\Phi(\overline{x}) \rightarrow P(\overline{x})) \rightarrow \forall\overline{x}(P(\overline{x}) \rightarrow \Phi(\overline{x}))$$

The inference relation $\vdash_P$ that results from circumscribing P is nonmonotonic. The above can be straightforwardly extended to circumscribe several predicates.

The model theoretic counterpart of circumscription is minimal entailment: A minimally entails q with respect to P, $A \models_P q$ if q is true in all models of A that are minimal in P[1]. McCarthy shows that if $A \vdash_P q$ then $A \models_P q$.

McCarthy notes that circumscription is not a nonmonotonic logic. He sees it as a form of nonmonotonic reasoning– jumping to conclusions– augmenting first order logic. The way one could use circumscription is for the reasoning system to apply circumscription to some predicates when the need presents itself, for example if the system has to make a decision that may be prevented by some other facts, it would want to minimize these obstacles to just those it knows about– minimizing the extent of the obstacles. Heuristics would decide when to circumscribe which predicates.

Formula circumscription, an extension of predicate circumscription was introduced in [108]. In the new version, formulas rather than predicates are circumscribed, and other formulas are allowed to vary during the minimization. Another difference is that now the circumscription is represented as a second order formula rather than as a axiom schema. Formula circumscription is defined as follows:

[1]Models minimal in P have the smallest extensions for P

Let $A(P)$ be a second order formula with P a tuple of predicate symbols in $A(P)$, let $E(P, x)$ be a well formed formula in which P and a tuple of individual variables, x occur free, then the circumscription of $E(P, x)$ with respect to $A(P)$, $A(P')$ is:

$$A(P') \wedge \forall P'[A(P') \wedge [\forall x E(P', x) \rightarrow E(P, x)] \rightarrow [\forall x E(P', x) \leftrightarrow E(P, x)]]$$

The predicates in $E(P, x)$ that are not in P are allowed to vary. Also the circumscription axiom is in the same language as the rest of the knowledge base and one can therefore reason about the axioms.

McCarthy also introduces the use of abnormality predicates to specify the use of circumscription in default reasoning. We can express that birds generally fly by $\forall x Bird(x) \wedge \neg ab(aspect1(x)) \rightarrow Flies(x)$. This says that if a bird is not abnormal in aspect1, then it flies. To reason about what birds fly, one circumscribes the abnormality.

To summarize, circumscription is a minimization approach to nonmonotonic reasoning: it minimizes the extent of certain predicates or formulas so we can jump to conclusions about them. It is expressed in second order logic for which there is no complete proof theory which makes implementation problematic.

7.8.2 Default Logic

Reiter [157] notes that inference made by default can be rejected by later information. These inferences are based on an absence of information to the contrary which Reiter equates to consistency. So, “if x is a bird, in the absence of information to the contrary, assume that x can fly” is taken to mean “If x is a bird, and it is consistent to assume that x can fly, assume that x can fly”.

Defaults are metarules of inference that are used to complete a theory. They are represented as:

$$\frac{\alpha(x) : M\beta_1 x, \ldots M\beta_m(x)}{\omega(x)}$$

where $\alpha(x)$ is the prerequisite and $\omega(x)$ is the consequence. The rule is read as: if $\alpha(x)$ and if it is consistent to assume $\beta_1 x, \ldots \beta_m(x)$, then conclude $\omega(x)$. A default theory is a pair (D, W) where D is a set of default rules, and W is a set of closed formulas.

The consequences of a default are given by the extensions of the theory which are defined using a fixed-point operator. Some default theories may have no extensions, while others may have several. Reiter

sees the purpose of default logic to be to provide an extension for the beliefs of a reasoner. An extension is to be used for reasoning until it is found that that is a bad extension. This leads Reiter to take a formula as “believed” as long as it is in any extension.

Reiter discusses normal default theories which are of the form $\alpha(x) : M\omega(x)/\omega(x)$. These have the interesting property of semi-monotonicity by which the proof theory is local with respect to defaults used in the proof, i.e., one does not have to consider all the defaults while building proofs. However, the proofs still depend on satisfiability tests for W. A further restriction is to consider closed normal formulas where all formulas in the default are closed. In this even more restrictive case, deciding whether a formula is present in the extension of a default theory is still undecidable.

Another view of the role of default reasoning is that of deriving new beliefs for a task: we define the default theory (D, W), determine if a given formula is derivable in the theory. If it is, that formula is added to W. In case new information is available that contradicts W, some sort of belief revision is needed.

In contrast to circumscription, default logic is a proof-based, fixed-point approach to nonmonotonic reasoning. Another difference is that the default in default logic is not accessible to the reasoner, whereas it is (although as a second order formula) in circumscription. The computational properties are not good since tests for satisfiability are needed for finding any extensions.

7.8.3 Autoepistemic Logic

Moore [130] reconstructs nonmonotonic logic [116] as a model of an ideally rational agent reasoning about its own beliefs to solve problems in nonmonotonic logic. The result is autoepistemic logic.

Moore denies that autoepistemic logic is default reasoning but says it is some kind of commonsense reasoning. Purely autoepistemic reasoning is not defeasible either. The nonmonotonicity of autoepistemic logic derives from the fact that the meaning of an autoepistemic statement depends on the theory in which it is embedded. The nonmonotonicity comes from the indexicality of the logic.

Moore uses a modal operator L that is taken to mean "is believed". Sets of autoepistemic formulas are interpreted as the beliefs of an agent. A belief $L\ P$ is true if and only if P is in the autoepistemic theory. The problem of inference is to determine what set of beliefs the agent should have based on some initial axioms.

The theory is taken to be sound and complete, resulting in an agent that is rational (because it is sound) and ideal (because it is complete). No real program can be this rational ideal agent, but Moore says they should approximate them.

Finding a syntactic characterization of an autoepistemic theory is done by fixed point definitions. No algorithms are forthcoming for computing the sets of formulas, but the sets can be defined. The following should be satisfied by an ideally rational agent:

1. If $P_1 \ldots P_m \in T$ and $P_1 \ldots P_m \vdash Q$ then $Q \in T$.
2. If $P \in T$, then $L\ P \in T$.
3. If $P \notin T$ then $\neg L\ P \in T$.
4. If $L\ P \in T$ then $P \in T$.
5. If $\neg L\ P \in T$, then $P \notin T$.

Conditions 1, 2 and 3 define stable autoepistemic theories: no further conclusions can be drawn from the theory. If a stable theory is consistent, then 4 and 5 also hold. These can be used to define stable expansions of a theory which are theories an ideal agent should believe. Just as in default logic, there can be several expansions of a set of beliefs, or none.

And just like default logic, autoepistemic logic is a fixed-point approach to nonmonotonic reasoning. However the sentences that give rise to the nonmonotonicity are in the language itself and can be reasoned about by the agent. One could argue that this is easier to do in autoepistemic logic than in circumscription. Just like default logic and circumscription though, the problem of inference is undecidable since the agent is omniscient (as can be seen from the characterization of stable expansions above).

7.8.4 Shoham's preferential models

[173] presents an attempt to provide an underlying semantical framework for non-monotonic logics. The approach is to consider the interpretations of a standard logic together with a preference relation between the interpretations. The aim is for different kinds of relations to produce the various non-monotonic logics known.

The relation $\subset$ is a strict partial order over the interpretations of the logic and $M_1 \subset M_2$ means that interpretation M_2 is preferred to interpretation M_1. An interpretation is said to preferentially satisfy a formula is that interpretation does satisfy the formula and there is no more preferred interpretation that does so. Based on this, preferential validity, satisfiability and entailment are defined.

This is used to describe circumscription and the modal approaches to non-monotonic logic. However, it does not model default logic appropriately.

Our intuition is that our system can be described using a preference relation over interpretations of a first order logic, but the definitions that Shoham uses do not seem to fit well. It seems more appropriate for our approach to say that an interpretation preferentially satisfies a formula if it satisfies the formula and there are no more preferred interpretations where that formula does *not* hold. Our view is that an interpretation will preferentially satisfy a formula if it is a most preferred interpretation and it satisfies the formula. If there is a more preferred interpretation that satisfies the negation of the formula, then the interpretation that does satisfy the formula does not preferentially satisfy it. This difference in the view of satisfaction makes our approach very different from Shoham's. However, more work needs to be done on this.

7.8.5 Truth maintenance systems

Truth maintenance systems (TMS) [43, 162] are to be used in conjunction with a problem solver and maintains the reasons for the beliefs of the system. The TMS does not itself make inferences about the domain–it determines the current set of beliefs and maintains the justifications for them. The beliefs, justifications and inferences are provided by the problem solver. TMS are meant to help in a wide range of roles including belief revision, contradiction handling, dialectical arguments, modeling others beliefs and more. We will focus on the belief revision and inconsistency control aspects here.

The TMS maintains beliefs as either IN or OUT. We can see these as being believed or not. A belief is IN if there is a reason for it. Note that there can be many reasons for a belief. The reason for a belief consists of two sets of beliefs, the in-list and the outlist. A belief is IN if all the beliefs in the in-list are IN and all the beliefs in the outlist are OUT. The first set of beliefs support the belief in question whereas the second set deny it. If any belief in the second set is IN, the belief in question is not.

These two sets allow different kinds of beliefs to be recorded:

- Normal deductions have the inlist non-empty and an empty outlist.
- Premises have both empty inlists and outlists.
- Assumptions have non-empty outlists. Assumptions depend, in part, on some other beliefs being OUT and these make the system non-monotonic.

The TMS allows the problem solver to add nodes which represent beliefs and to add or delete justifications and to mark some nodes as contradictory. Once the problem solver adds or deletes justifications of a node, the TMS process makes the appropriate changes to the set of beliefs–making them IN or OUT as need be. In case a contradictory node becomes IN, the TMS backtracks to find assumptions on which the contradiction depends. It then picks one of them to remove to regain consistency.

The TMS is different from the other formalisms we have seem in that it does no reasoning but it allows for belief revision and contradictions which are two crucial aspects of logics for agents. Our CR rule, just like the TMS depends on the structure of the derivations for resolution of contradictions. However, instead of being in a separate process, it is part of the reasoning system. Since the CR rule depends only on the form of the derivations, it seems that the same functionality could be obtained from a TMS linked to a reasoner.

However there are some crucial differences between the two approaches:

- The reasoner needs to explicitly flag some nodes as contradictory. The TMS does not represent and is not sensitive to the logical relations between beliefs. So it cannot on its own detect inconsistencies. The number of possible inconsistencies is large and it is not practical for each possible contradiction to be noted in the TMS.

- It is not clear how the TMS would react to preferences between defaults. While default could be represented as assumptions, it is not clear how one would represent the preference between classes of assumptions. It seems that the TMS would need to have access to some domain information and the algorithms would need to be changed for that information to be taken into account.

The TMS approach and features are quite similar to our own, but the disconnect between the TMS and the reasoner make it unsuitable for our purposes. In our case, even though the reasoning we do in case of contradictions is purely procedural, the fact that the reason maintenance is done in the same system as the reasoner is a great advantage. This will be even more apparent if the constraints we have on the reasoning are relaxed as we do later.

7.8.6 Delgrande's conditional logic

Delgrande has developed default formalisms based on conditional logic in which, among other things, attention is paid to the representation of the denial of defaults (see [40] and [41]). In these logics—there are two of them, one propositional and one first-order—the conditional operator $\Rightarrow$ is added to an otherwise standard logical language in order to represent defaults, or what Delgrande calls statements of "normality." In the propositional version (called **NP**) a default is a statement of the form $\alpha \Rightarrow \beta$; intended to be read roughly as "if α then normally β"; for example: "if it is raining then normally the grass is wet." In **N** (the first-order version) defaults are written in the form $\forall x(\alpha \Rightarrow \beta)$, intended to be read as "α's are normally β's," for instance: "ravens are normally black." A possible world semantics is the basis for truth in both **NP** and **N**. Specifically $\alpha \Rightarrow \beta$ is true if β is true in each of the least exceptional worlds in which α is true, and $\forall x(\alpha(x) \Rightarrow \beta(x))$ is true in a world *w* iff $(\alpha \Rightarrow \beta)$ is true for each individual in the domain of individuals in *w*.

Delgrande's primary aim in these logics is a mechanism for reasoning *about* defaults (for drawing conclusions which themselves are defaults), rather than reasoning *with* them (e.g., for drawing conclusions about individuals). As examples, in the propositional case, the default $P \Rightarrow Q$ follows from $P \rightarrow Q$, and in the first-order case $\forall x(P(x) \Rightarrow R(x))$ follows from $\forall x(P(x) \Rightarrow Q(x))$ and $\forall x(Q(x) \rightarrow R(x))$.

Indeed, the first-order version, **N**, does not permit default reasoning *about* individuals. This is due to the fact that despite the suggestive notation, the default $\forall x(\alpha \Rightarrow \beta)$ is not to be construed as concerning ordinary quantification. Specifically, no mechanism is provided for substituting individuals for x, nor therefore for drawing default conclusions about an individual x from the default. Thus, though intuition may suggest that $Flies(tweety)$ ought to follow from $\forall x(Bird(x) \Rightarrow Flies(x))$ and $Bird(tweety)$, if Tweety is "normal", such is not the case in **N**; there simply is no inference rule nor axiom (schema) in the logic to help accomplish this. Delgrande provides a separate mechanism for drawing default conclusions about individuals.

One interesting feature of Delgrande's approach is the relationship which arises between typicality entailment ($\Rightarrow$) and strict entailment ($\rightarrow$): $\forall x(Px \Rightarrow Qx)$ follows from $\forall x(Px \rightarrow Qx)$. One could argue whether this is appropriate. Certainly $\Rightarrow$ and $\rightarrow$ bear some relation to one another, as Delgrande notes, though just what that relation ought to be may not be clear. On the one hand, it may seem reasonable that if *all P's are Q's* then *typically P's are Q's*, as comes out in Delgrande's formalism. On the other hand it can be considered misleading to assert that consequent. For instance asserting that *typically birds are animals* is surely misleading as it encourages the listener to assume that some birds are not animals. E.g., from the assertion that *typically birds are animals* Grice's [72] maxim "be maximally informative" would lead to the inference that most, but not all, birds are animals. Also, [51] argue that it is in the very nature of a default to have counterexamples.

7.8.7 Poole: default reasoning as theory formation

Poole [149] bases his work on the premise that reasoning is not a matter of deduction but of theory formation. He views default reasoning as an attempt to use a set of hypotheses and a set of facts to explain observations. The hypotheses used in the explanation play the role of defaults.

Poole uses a first order language together with a set of facts $\mathcal{F}$, a set of names of possible hypotheses Δ, and a set of constraints C. Names of possible hypotheses are predicates of the same arity as the hypotheses they name. For each name in Δ there is a fact in $\mathcal{F}$ which links the name to the hypothesis. If, for example, we want to have as default that birds normally fly, we would put $birdsfly(x)$ in Δ as the name of a hypothesis. $\mathcal{F}$ would then contain $birdsfly(x) \rightarrow (bird(x) \rightarrow fly(x))$. The constraints in C are formulas that state when a particular hypothesis is not applicable. For example, if we do not want the $birdsfly(x)$ hypothesis to apply to ostriches, we would put $ostrich(x) \rightarrow \neg birdsfly(x)$ in C. Poole defines a scenario of $\mathcal{F}$, Δ, C to be a set $D \cup \mathcal{F}$ where D is a set of ground instances of Δ and $D \cup C \cup \mathcal{F}$ is consistent. An explanation of a closed formula g is then a scenario that implies g.

As an example, let $\mathcal{F} = \{bird(tweety), ostrich(fred), birdsfly(x) \rightarrow (bird(x) \rightarrow fly(x)), ostrich(x) \rightarrow bird(x)\}$, $\Delta = \{birdsfly(x)\}$, and $C = \{ostrich(x) \rightarrow \neg birdsfly(x)\}$. With this, we can explain $fly(tweety)$. The set D contains $birdsfly(tweety)$ which lets us derive $bird(tweety) \rightarrow fly(tweety)$ from $\mathcal{F}$, and from that we can get $fly(tweety)$. However we cannot find an explanation for $fly(fred)$ because $ostrich(fred)$ and the hypothesis $birdsfly(fred)$ are inconsistent with the constraint in C.

Unlike the approaches of Delgrande and Schlechta (see below), Poole's treatment combines reasoning *with* defaults with reasoning *about* defaults.

7.8.8 Schechta's generalized quantifiers

Schlechta [170] represents defaults in first order logic with a generalized quantifier. An open default such as "Normally birds fly" is taken to mean "Most birds fly" and is represented as $\nabla x bird(x) : fly(x)$ which says that there is an "important" subset of the set of birds, all of whose members fly. We will mainly discuss the simpler case of normal open defaults: "Normally things don't fly" which is represented as $\nabla x \neg fly(x)$. This says that there is an important subset of the set of all objects, all of whose members don't fly.

The notion of important subsets is captured in the semantics by a system $\mathcal{N}(M)$ of important subsets of the domain. $\mathcal{N}(M)$ consists of subsets of M such that the intersection of any two of the subsets is not the empty set (unless the domain is empty). This condition rules out contradictory important subsets and therefore rules out situations where both "normally $\phi(x)$" and "normally $\neg\phi(x)$" are true. Also ruled out are empty important subsets when the domain is not empty. The notion of truth follows that in first order logic with additional inductive steps including: $\nabla x \phi(x)$ holds in $\langle M, \mathcal{N}(M) \rangle$ provided there is a set A in $\mathcal{N}(M)$, for all members a of which $\phi(a)$ holds. This captures the idea that for something to be normally true there has to be an important subset of the domain for which that property holds. Since it is not possible for an important subset of the domain to satisfy $\neg\phi(x)$ while $\forall x \phi(x)$ holds, we cannot have "peaceful coexistence" between $\nabla x \neg\phi(x)$ and $\forall x \phi(x)$.

The language is a first order language augmented with the quantifier ∇ and with a set of axiom schemata. The equivalence property is obtained from the axiom schema: $\nabla x \phi(x) \wedge \forall x(\phi(x) \rightarrow \psi(x)) \rightarrow \nabla x \psi(x)$. Inconsistencies in the defaults can be detected using the axiom $\nabla x \phi(x) \rightarrow \neg\nabla x \neg\phi(x)$. Analogous axioms are present in the open default case where another axiom of interest is $[\nabla x \phi(x) : \psi(x)] \wedge \forall x(\phi(x) \wedge \psi(x) \rightarrow \vartheta(x)). \rightarrow \nabla x \phi(x) : \vartheta(x)$. This enables us to go from "Normally birds fly" and "Things that fly have wings" to "Normally birds have wings". However we also have: $\forall x \phi(x) \rightarrow \nabla x \phi(x)$ which (as mentioned earlier in the discussion of Delgrande) we find unintuitive: if all birds are animals, then it seems misleading to say that typically birds are animals.

This logic was developed in the context of an order sorted system where defaults are attached to the sorts. In case of inconsistencies between defaults or between defaults and certain other information, this order can be used to select a consistent subset of these defaults. The general rule is that more specific information is preferred over more general information.

Schlechta's approach enables us to reason about (and deny) defaults and to resolve conflicts between defaults in a principled way. But as with Delgrande's first order proposal, we cannot use Schlechta's scheme to conclude facts about individuals. There is no axiom or inference rule that lets us go from $\nabla x \neg fly(x)$ to $\neg fly(fred)$.

7.8.9 Approximate inference

In [25], Cadoli and Schaerf describe how default logic and circumscription can be approximated for the propositional case. The approximation is based on their earlier work on approximating propositional logic [23]. They define two consequence relations, one of which is incomplete and the other unsound with respect to propositional inference.

The inference relations are based on interpretations which in one case map all propositions except those in a set S into the truth value 0, and in the other case map all propositions except those in a set S onto truth values 0 or 1, but so that not both p and $\neg p$ are mapped onto 0. The inference relation based on the first interpretation is complete but not sound whereas the second is sound but not complete. By expanding the set S, the approximate inference relations converge on the correct consequence relation $\models$. These consequence relations are used to define approximations to credulous default reasoning assuming the defaults are normal defaults in CNF and there are no tautologies. Two consequence relations are defined in this case too, one unsound and the other incomplete with respect to credulous default reasoning. These converge to the correct answer as the set S is expanded. The approach to circumscription is similar.

The main differences between this approach and ours is that this approach has a well defined semantics and a well defined notion of the approximation of the logics. However, the logics treated are propositional and not first order as our logic and there are more restrictions on the form of the logic in the case of Cadoli and Schaerf. The propositional default logics, while being intractable are not undecidable, so that there is a point at which all the computation stops and there is a final answer. The first order case, however is is undecidable and we cannot know that we have the right solution.

7.8.10 Ghose and Goebel

Ghose and Goebel [65] consider propositional, prerequisite-free, semi-normal defaults for their approach to anytime default reasoning. They consider two approximations that are relaxed to approach the correct solution. α-approximations consider a subset of the propositions in the default theory. β-approximations on the other hand, consider a subset of the default theory. The problem of computing these partial solutions is then mapped onto partial constraint satisfaction problems. Different techniques correspond to the α and to the β partial solutions. Algorithms for the partial constraint satisfaction are then used to compute partial solutions to the default inference.

Similar to the above, this approach is restricted to propositional default logic and further restricts these to prerequisite-free semi-normal defaults. The same remarks about propositional defaults apply.

7.9 Future work

The logic and the rules of inference above have been motivated by appealing to intuition and to the behavior of the presupposition and the implicature reasoning systems. We have tested our non-monotonic system with a number of examples in which it behaved correctly. However, this is not sufficient for characterizing the logic. We need to specify a semantics for the logic and find what relation the rules of inference given above have with these semantics. Other work involves extending the logic so that it can reason with preferences and use more complex diagnoses of contradictions.

7.9.1 Characterizing the logic

The semantics we propose to work out are a possible-worlds semantics with defaults and preferences corresponding to preferences between sets of worlds. Note that our approach differs from that in [173] in the definition of entailment and satisfiability.

We take a non-monotonic theory to consist of a first order set of sentences, a set of defaults and a set of preferences among the defaults. The defaults and preferences in the theory induce a preference relation among the interpretations of the first order part of the theory. A default is said to hold, be compatible or not be compatible with an interpretations according to whether the antecedent and consequent are true, or the antecedent is false, or the antecedent true and the consequent false in that interpretation. An interpretation is preferred to another according to the status of defaults holding or being incompatible in the interpretations and on the preference between these defaults. This preference relation is a partial order on the interpretations of the first order part of the theory and we take a formula to be preferentially entailed by the model if it is true in all the most preferred interpretations in the model. Other semantic properties follow from this.

The inference rules would ideally assert in the KB only and all those formulas true in all the most preferred interpretations. The problem though is that they jump to conclusions before all relevant information is known. That is where contradictions occur and formulas need to be retracted from the KB.

If there are no IDs (sets of inconsistent defaults) in the theory, then all the conflicts between defaults are arbitrated by default preferences. Further, from the rules we presented, the only contradictions that occur are those in which a single default is inconsistent with the first order part of the theory. Therefore any misapplication of rules will be correctly repaired when they are found. If we assume that the inference rules are applied fairly, then we eventually get to the correct default applications in the KB. Once this has happened, the state of that default will not change. Therefore, the KB will eventually stabilize with the correct defaults holding. The logic may derive wrong answers intially, and may switch from bad to good and back to bad answers several times, but it will eventually get to the correct answer and not change anymore.

Further, if we assume some probability distribution on the minimun lengths of proofs for those formulas that are of interest, and a distribution on the depth of the default preference trees, we can derive the probability that the correct answer for a formula of interest has been derived after a given amount of computation. This probability can be used as a measure of goodness for the answer the logic produces resulting in the logic being an anytime algorithm [16].

In case there are IDs, a potential problem is that our method to identify the IDs may include defaults that do not belong to the ID. This fact can be discovered after more computation, but there is no guarantee that it will not recur. A possible behavior then is that a default may be taken to hold and not to hold and so on indefinitely. We currently see two approaches to this problem: (1) to change the inference rules; (2) to restrict the application of inference rules so that this does not occur. The second solution may restrict the possible strategies of control needed for greater efficiency. This is however also likely to result in incompleteness.

7.9.2 Extending the logic

We can also extend the logic to allow reasoning about defaults and preferences. There might be circumstances where we need to reason about defaults to decide which to prefer, or to derive new defaults. We might also want to reason with preferences. These are not part of the current logic where we assume that all defaults and preferences are available at the start of reasoning. In more realistic settings (see later), this might not be possible and we will need to do the sort of reasoning mentioned above. The difficulty then is that the algorithms for applying defaults and solving contradictions will not be suitable anymore because we might later find some default or some preference that we did not know about earlier. This introduces a new class of mistakes that needs to be taken care of. It is not clear whether there can then be simple inference rules to do so as is the case now.

7.9.3 More complex diagnoses

The complications introduced then might also require a more sophisticated diagnosis of the problems when contradictions are encountered. Currently, the diagnosis is compiled into the inferences to apply defaults and handle contradictions. But in the new setting, this might not be sufficient and the logic may have to reason explicitly about mistakes to diagnose contradictions. All of these new features can be represented and implemented in Alma, so there is no obstacle as far as implementation is concerned.

7.10 Conclusion

We started by considering a generalization of the natural language applications in the previous chapters. This led us to the use of a non-monotonic language to describe the mistake handling behavior of the generalized system. We then specified the rules of the system and we considered its use as a non-monotonic reasoner.

We note that the reasoning system we described is suited for kinds of mistakes in beliefs rather than for all mistakes. This restriction to mistaken beliefs allowed us to compile the responses to the mistakes in inference rules. In more complex situations when we need to reason about actions and intentions, this approach might not be applicable.

We briefly considered how this system could be used as a non-monotonic reasoner. The characteristics of the system differ from the usual non-monotonic logics in several ways. The intuition is that our system has an anytime behavior–the answer improves over time. This leads to future work to formalize this intuition. The semantics can be a formalization of the informal semantics discussed in this chapter. The goodness measure that we need for showing anytime behavior may be a measure of the probability that the answer we have at one point is going to change. This may depend on some properties of the domain and these restrictions may themselves be of interest.

Chapter 8

Non-monotonic reasoning in Alma

The aim of the rules and representations of the previous chapter were to have a practical system that generalizes the treatment of mistakes in the presupposition and implicature applications by specifying this behavior in a non-monotonic language. The implementation of this system in Alma is described in this chapter. The system was implemented as an Alma application, without modifying any the Alma algorithms. This preserves the generality of Alma by not committing Alma to any particular approach to non-monotonic reasoning, but it also makes the resulting logic less transparent and less easy to use.

In the next section we describe the representations and procedures that were used to implement the logic. These are approximations of the algorithms presented in the previous chapter and we discuss these differences in the subsequent section. These approximations make the implementation easier and the run-time complexity lesser at the cost of generality. The section after that illustrates the implementation with various behaviors involving defaults. We conclude with future work to be done.

8.1 Implementation of the rules in alma

In this section we describe how the DA (default application) and CR (contradiction resolution) rules described in the previous chapter are implemented in Alma. With the appropriate sentences in the logic representing defaults and preferences, these rules allow the logic to deal with a large variety of domains (with mistakes) in a uniform way.

The approach was to implement the rules without modifying the Alma system. The representations and procedures are built on top of Alma primitives rather than being themselves Alma primitives. This makes use of the ability of Alma to use new representations and Prolog procedures. Some consequences of that are that the representations are not as economical as they could have been; the procedures to apply the rules take place over more than one step and the representations and procedures are not very transparent. Applying a default, for instance, takes a few steps with intermediate data being asserted in the Alma KB.

We first describe the representations used in the implementation and then the procedures that implement the DA and CR rules.

8.1.1 Representation

The implementation of the rules above require several representations that are presented here. These are predicates in the Alma language and should be considered to be reserved predicates.

Defaults

Defaults are written as `fif` formulas with names that have to be parameterized by by the variables used in the default. This facilitates finding instantiations of the defaults. The fact that the formula is a default has to be stated by a separate `default` formula.

The default $\forall x\ P(x) \hookrightarrow Q(x)$ is represented as:

```
named(fif(P(x), conclusion(defcon(d1(X), Q(x)))), d1(X)).
default(d1(X)).
```

The fact that the formula is a default has to be asserted explicitly in the KB since there is no other way to distinguish defaults from non-defaults.

The default itself is represented as a `fif` formula so that, among other things, the default is not contraposed. The conclusion of the `fif` is a complex term that includes the name of the formula itself as its first argument and the conclusion desired as the second.

defcon

`defcon(N, C)` represents that the DEFault CONclusion of the default named by the first argument is the second argument. The latter is the conclusion of the default and is a formula that may be eventually added to the KB. This allows the logic to associate directly in the KB the names of the default instances to the conclusions obtained. It is important that the name be the name of the instance of the default rather than that of the quantified default. The logic uses this trick to obtain the name of the instance of the default since Alma does not have an instantiation rule (or a universal quantifier elimination rule). This is important in case of preferences as we will see later. Applying the `fif` inference rule to a default does not therefore result in the conclusion being asserted but only in the assertion of what a potential conclusion of the default could be. This corresponds to verifying that the premise is in the KB. After that we need to verify that the default is applicable.

applied_default

If it is found that a default can be applied, that fact is asserted in the KB through a `applied_default(N, C)` formula. The arguments are identical to the corresponding `defcon(N, C)` formula. This signals to the logic that `C` can be added to the KB.

add_default_conclusion

Once the conditions have been verified, and before the conclusions of the default are to be added to the KB, defaults that are less preferred than the one in question have to be removed. This occurs after `applied_default(N, C)` is asserted in the KB. Once this is done, `add_default_conclusion(N, C)` is asserted in the KB. This indicates that the conclusion `C` can be safely added to the KB now.

prefer

Preferences are represented as would be expected, using the names of the defaults. The fact that the names of defaults are parameterized with the variables of interest in the default gives some flexibility in stating preferences.

The usual type of preference one would like to assert is something of the sort "Prefer an instance of the application of the default that birds fly to the same instantiation of the default that animals don't fly". Assuming that the names of the defaults are `birdsFly(X)` and `animalsDontFly(X)` this is represented as:

```
prefer(birdsFly(X), animalsDontFly(X)).
```

If we wanted to express that the fact that any bird flies is to be preferred to any animal not flying (even an animal that is different from the bird), we could do this as `prefer(birdsFly(X), animalsDontFly(Y))` In this case, once a `birdsFly(X)` default has been applied to some bird, no instance of `animalsDontFly(X)` can be used since that application of `birdsFly` is preferred to it.

If we have a preference that applies to just one object in the domain, we can express it as `prefer(birdsFly(joe), animalsDontFly(joe))` for instance.

default_apps

When there is a contradiction, one of the actions the logic takes is to compute the defaults that the contradictand depends on. There can be several sets of such defaults for each derivation of that contradictand. This is represented in the KB by

```
default_apps(N, D)
```

`N` is the name of the contradictand formula and `D` is a list of lists of defaults, each list being the defaults used on one derivation of `N`.

depends_on_defaults

The logic also asserts whether the contradiction has a derivation that does not depend on any default. This is easily computed by verifying that one of the elements of the list `D` in `default_apps(N, D)` is the empty list. This is expressed in the KB as

```
depends_on_defaults(N)
```

distrust

Alma automatically distrusts formulas when there is a contradiction. These are removed from the list of formulas that can be used for the inference rules and a `distrust(n1)` formula is added to the KB where `n1` is the name of the formula to be distrusted. The procedures implementing the CR rule need to distrust formulas when the cause of a contradiction has been located to a default. The distrust is done by a Prolog procedure which also asserts the appropriate formulas in the KB.

8.1.2 Procedures

The algorithms used to implement the rules described above use the ability of Alma to run Prolog programs as part of inference rule applications. This makes it much easier to implement the algorithms required than to do them a step at a time in Alma. There are two main procedures: one to apply defaults (DA) and one to handle contradictions (CR), which are the only mistakes that we consider here. These procedures are split into a few smaller procedures that store intermediate results in the KB using the representations described above.

DA—default application

Step 1
Given a default

```
named(d1(X), fif(p(x), conclusion(defcon(d1(X), q(x))))).
default(d1(X)).
```

The usual Alma rules apply and once the antecedent of this formula (`p(X)`)is present in the KB [1], the conclusion is asserted. If `p(a)` is present in the KB, we get:

```
defcon(d1(a), q(a)).
```

This asserts that `q(a)` is a potential default conclusion of `d1(a)`.

Step 2
We now need to verify that there is nothing known at that time that would stop `q(a)` from being added to the KB. This is done by:

1. Gathering all the formulas of the form `prefer(N, d1(a))`. This is done with the `gather_all` predicate of Alma. `gather_all(X, L)` searches for formulas that unify with `X` that are in the KB and returns them in the list `L`.
2. Verifying that there are no more preferred default than $q1(a)$ that have been applied. Given the list of `prefer(N, d1(a))` gathered in the previous step, the logic searches for applications of the preferred defaults. These are represented in the KB as `applied_default(N, C)` and can be simply looked up.
3. If there are none, this default can be applied and `applied_default(d1(a), q(a))` is added to the KB.

Step 3

At this point the less preferred defaults whose consequents are inconsistent with the consequent of `d1(a)` are in the KB, if there are any. These were mistakenly applied and should be removed before the consequent of `d1(a)` is added. Doing this preempts a future contradiction with `q(a)`. This is done as follows:

1. Gather all the defaults less preferred than `d1(a)`. This is done as above using `gather_all`.
2. For each of these defaults:
   (a) Find their applications through `applied_default`.
   (b) Delete the `applied_default` and undo its consequences. The `defcon` formula corresponding to the less preferred formula is left in the KB because that default is still potentially applicable, but since the conditions for its application are now false (since a more preferred default has been added), the `applied_default` formula and its consequences are removed.
3. Assert that this has been done by asserting `add_default_conclusion(d1(a), q(a))`.

Step 4
At this point all that is left is to add `q(a)` to the KB. This concludes the application of a default. Note that this procedure takes four steps and adds intermediate results to the KB.

[1]More accurately, as soon as each element of the antecedent is present in the KB

CR—contradiction resolution

The detection of contradictions is done automatically by Alma as soon as it occurs and a formula of the form `contra(P, Q, T)` is asserted where `P` and `Q` are the names of the contradictands and `T` is the step number at which the contradiction was detected. Alma also distrusts the contradictands and their consequences. The assertion of `contra(P, Q, T)` is the trigger for the following procedure to resolve the contradiction.

Step 1

1. The leaf defaults in each derivation for each contradictand are computed. This is easily done through a Prolog program that traverses the derivation tree for the contradictand. The derivation is obtained through the meta-relations in Alma.
2. These defaults are asserted in a `default_apps` formula for each contradictand as seen above.

Step 2
At the next step, `depends_on_default` is computed. If a contradictand `C1` has a derivation that depends on no default, then `not(depends_on_default(C1))` is added to the KB, else `depends_on_default(C1)` is added. This is easily computed by verifying whether `default_apps(X, Y)` contains an empty list in `Y` for contradictand `X`.

Step 3
Then the logic attempts to resolve the contradiction. As seen earlier, because the DA rule removes less preferred defaults when any default is applied, and does not apply any default if a more preferred one is applied, there are no contradictions between defaults that are related by the preference relation. The only possibilities then are that one of the contradictands was derived non-defeasibly (and `depends_on_defaults` is false for it, or that there is an ID. This is done in the following way:

1. If `depends_on_defaults` is false for one of the contradictands, then for each set of defaults the other contradictand depends on,
   (a) If it is a singleton set, remove the corresponding `applied_default` and undo its consequences. This is a case of a default being inconsistent with a non-defeasible formula and the default application being mistaken.
   (b) If it is a set of defaults, this is an ID and distrust the `applied_default` corresponding to the defaults and their consequences too. Here we cannot choose between the defaults applied and the solution is not to apply any of them.
2. If both contradictions depend on defaults, then
   (a) Compute the cross-product of the lists in the `default_apps` formulas.
   (b) For each set of defaults
      i. If it is a singleton set, remove the corresponding `applied_default` and undo its consequences.
      ii. If it is a set of defaults, this is an ID and distrust the `applied_default` corresponding to the defaults and their consequences too.

We see that defaults are applied when it seems from the state of the KB, that they can be added. However, as soon as evidence is discovered that they have been mistakenly applied, the default application is removed.

8.2 Comparison with the rules

We compare the implementation we describe above with the algorithms expected for the rules of the previous chapter. The implementation makes some assumptions and simplifications to keep the procedures manageable. These have not had serious adverse effects in our tests. However it is possible to construct cases where they will fail.

- **No complete FOL**

  We assumed that we could derive all possible formulas (except for those derivable from contradictions) from the underlying logic. This is not the case here. Alma typically runs in the forward direction with resolution, therefore we lose the completeness that is assumed. We may therefore not derive some important formulas.

  Assuming that the domain description consists of defaults, implications and atomic formulas, this is not a problem. If there are disjunctions however, some conclusions might not be derivable. For instance if $\alpha_1 \wedge \alpha_2$ and $\alpha_1 \hookrightarrow \beta$ and $\alpha_2 \hookrightarrow \beta$ are in the KB, we would expect to derive β. However this is not derivable in the current system.

- **No explicit representation of default instances**

  The procedures above do not instantiate defaults as a separate step before applying them. This makes it less straightforward to identify the default instances used to derive formulas of interest. This is a result of the use of resolution as the main rule of inference in Alma. To get around this problem, we use the `defcon` representation to relate the defaults instances to their consequences. This solves the problem, but is rather inelegant.

- **No explicit representation of IDs**

  The implementation does not explicitly represent IDs. This facilitates the implementation and speeds up the application of defaults and other reasoning processes but also has some problematic consequences. The solutions to these problems is provided by the control strategy Alma currently uses. If this is changed, the assumptions that solve the problems will no longer be valid and the IDs may have to be explicitly represented. The consequences of not representing IDs are:

  – **Defaults not verified for ID membership** According to the rules in the previous chapter, a default must be checked for membership in an ID before applying it. Since we do not explicitly represent IDs here, this can't be done and potentially allows the logic to derive formulas it should not.

    This is not a problem though, since the control of the logic applies an inference rule to a set of premises just once. If δ_i say, is found to be in an ID, then the premise of δ_i and δ_i itself will have already been used once. That would have eventually resulted in the ID being found, the consequences will have been distrusted and this instance of δ_i will not be applied any more so that there is no risk of a future mistaken application of δ_i.

  – **No ID subset verification**

    According to the CR rule in the previous chapter, if $\delta_i, \ldots, \delta_j$ are jointly inconsistent, the logic has to verify whether there is a subset of these defaults that is an ID. If there is, these are not minimal and therefore not an ID. This cannot be done here since the IDs are not explicitly represented and the logic runs the risk of not applying a default that should be applied.

    This problem does not appear since the control strategy amounts to a breadth-first exploration of the formulas derivable. In that case, the extra defaults that can cause non-minimal sets of inconsistent defaults are not likely to be present in the derivations of inconsistency. This is still a possibility of that although we have not encountered it in our tests.

  – **No proofs for ID subsets**

    For the same reason as the above, the logic does not verify whether some subset of the presumed IDs it derives is inconsistent. Since we assume that we do not get superset IDs, this is not necessary.

8.3 Illustration of behavior

We illustrate the behavior of the system with a few simple cases of nonmonotonic reasoning.

8.3.1 Simple default application

We have a default whose antecedent is in the KB and we show the steps before the consequent is asserted in the KB.

The KB initially contains the default and the antecedent:

```
named(fif(bird(X), conclusion(defcon(birdsFly(X), flies(X)))), birdsFly(X)).
default(birdsFly(X)).

bird(tweety).
```

together with the Alma formulas describing the behavior of the system for nonmonotonic reasoning which we don't show.

At the first step, the relevant new formula is:

```
defcon(birdsFly(tweety), flies(tweety)).
```

The default has been applied and the consequence of the Tweety instantiation of the default is that Tweety flies. At the next step we get:

```
applied_default(birdsFly(tweety), flies(tweety)).
```

The logic has verified that there is no obstacle to applying this default and is next going to remove weaker defaults. This is done in the next step and the new formula is:

```
add_default_conclusion(birdsFly(tweety), flies(tweety)).
```

The default is asserted at the next step:

```
flies(tweety).
```

The KB does not change subsequently apart from incrementing the step number.

8.3.2 Default application preempted by non-defeasible information

This is similar to the above case, but this time, the negation of the consequent is present in the KB. This prevents the application of the default. The KB this time round contains in addition the fact that Tweety does not fly:

```
named(fif(bird(X), conclusion(defcon(birdsFly(X), flies(X)))), birdsFly(X)).
default(birdsFly(X)).

bird(tweety).
not(flies(tweety)).
```

We still obtain the assertion that a possible consequence of the default is that Tweety flies:

```
defcon(birdsFly(tweety), flies(tweety)).
```

And that is all that is derived. Because of the presence of non-defeasible information that Tweety does not fly, the default cannot be applied and the derivation does not progress beyond that point. In particular, `applied_default(birdsFly(tweet` does not get added.

8.3.3 Default application preempted by a preferred default

This time, there are two defaults, one of which is preferred to the other. Only the preferred one is applied. In this example, we have two defaults, one which is the same as before and the other a default stating that penguins typically don't fly. We also have a preference for the second default to the first.

```
named(fif(bird(X), conclusion(defcon(birdsFly(X), flies(X)))), birdsFly(X)).
default(birdsFly(X)).

named(fif(penguin(X), conclusion(defcon(penguinsDontFly(X), not(flies(X))))),
      penguinsDontFly(X)).
default(penguinsDontFly(X)).

prefer(penguinsDontFly(X), birdsFly(X)).

if(penguin(X), bird(X)).
penguin(tweety).
```

We also know that Tweety is a penguin and that penguins are birds. At the first step, we derive that Tweety is a bird and also that a consequence of the penguinsDontFly default is that Tweety does not fly:

```
defcon(penguinsDontFly(tweety), not(flies(tweety))).
bird(tweety).
```

Since there is nothing stopping the application of the penguins default, that proceeds and at the same time the logic asserts that a possible conclusion of the birdsFly default is that Tweety flies:

```
applied_default(penguinsDontFly(tweety), not(flies(tweety))).
defcon(birdsFly(tweety), flies(tweety)).
```

At the next step, two things happen: the logic tries to remove any default that has been applied that is less preferred than the penguin default applied to Tweety, and the logic verifies whether there is a default that has been applied that is preferred to the birds-fly default applied to Tweety. Since the logic has started to apply the penguinsDontFly default to Tweety, the birds-fly default does not apply and the other default continues the process of being applied:

```
add_default_conclusion(penguinsDontFly(tweety), not(flies(tweety))).
```

This results in Tweety not flying:

```
not(flies(tweety)).
```

And no new formulas are derived except for the clock.

8.3.4 Detection of IDs

This time the logic detects that there are two defaults that are inconsistent and for which it does not have preferences. This results in an ID. The initial set of formulas is as above except that we do not have the preference between the defaults. Also we assert both that Tweety is a bird and that it is a penguin so that we do not have a delay in the application of the defaults. This does not change the end result of the computation. The initial KB is:

```
named(fif(bird(X), conclusion(defcon(birdsFly(X), flies(X)))), birdsFly(X)).
default(birdsFly(X)).

named(fif(penguin(X), conclusion(defcon(penguinsDontFly(X), not(flies(X))))),
      penguinsDontFly(X)).
default(penguinsDontFly(X)).

if(penguin(X), bird(X)).
penguin(tweety).
bird(tweety).
```

At the first step, both defaults become applicable and we have:

```
defcon(birdsFly(tweety), flies(tweety)).
defcon(penguinsDontFly(tweety), not(flies(tweety))).
```

Since neither has a more preferred default that has been applied, we move on to the next step:

```
applied_default(penguinsDontFly(tweety), not(flies(tweety))).
applied_default(birdsFly(tweety), flies(tweety)).
```

There is nothing to delete here and we go to the next step:

```
add_default_conclusion(penguinsDontFly(tweety), not(flies(tweety))).
add_default_conclusion(birdsFly(tweety), flies(tweety)).
```

This results in both defaults being applied and we get:

```
flies(tweety).
not(flies(tweety)).
```

This is inconsistent and the direct contradiction is recognized at the next step:

```
contra(30, 29, 5).
distrusted(30, 5).
distrusted(29, 5).
```

Here 29 and 30 are the names of the contradictory formulas and 5 is the time at which the contradiction has been detected. in addition to asserting that the contradictands have been distrusted, they are in fact distrusted also. We now get into the contradiction handling procedures. The first thing is to compute whether the contradictands depend on defaults, we get:

```
default_apps(29, [[23]]).
default_apps(30, [[24]]).
```

This says that there is just one derivation for formula 29 and default 23 is in that derivation, and similar information for default 30. The result is that both contradictands depend on defaults:

```
depends_on_default(29).
depends_on_default(30).
```

So that neither contradictand can be trusted and we go ahead and distrust the consequences of the defaults. Note that when we get the contradiction we distrust the consequences of the contradictands, but here we distrust the consequences of the default itself which might have been applied several inference steps earlier and from which other formulas might have been obtained and will now be distrusted.

```
eval_bound(distrust_applied_defaults(29, [[23]]), [[[23]]]).
eval_bound(distrust_applied_defaults(30, [[24]]), [[[24]]]).
```

The consequences of these are located and distrusted and nothing new is derived other than the assertions that these consequences have been distrusted which we do not reproduce here. Therefore the logic has no opinion as to whether Tweety flies or not.

8.3.5 Default application removed by a non-defeasible formulas

In this case, the default applies and after that, non-defeasible information is available that contradicts the default. The default is then removed. This case is similar to the corresponding case above except that the fact that Tweety does not fly is added later. Just as the first case, the logic asserts that Tweety flies from the birdsFly default:

```
flies(tweety).
```

If we now add that Tweety does not fly, we get a contradiction as above. But this time, (after a few steps):

```
default_apps(22, [[18]]).
default_apps(27, [[]]).
```

The non-defeasible formula (27) does not depend on any default. This is asserted at the next step:

```
depends_on_default(22).
not(depends_on_default(27)).
```

In this case, the fact that Tweety does not fly is reinstated and we go on to distrust the consequences of the default instance that birds fly applied to Tweety:

```
eval_bound(distrust_applied_defaults(22, [[18]]), [[[18]]]).
not(flies(tweety)).
```

After distrusting the appropriate formulas, nothing changes and the logic contains the fact that Tweety does not fly.

8.3.6 Default application removed by a preferred default

This is similar to the above case except that this time a more preferred default is applied which removes the first default. We do this just like the above preference case except that we assert that Tweety is a penguin after we have derived that it flies instead of doing so in the initial KB.

After applying the first default, we get:

```
flies(tweety).
```

We then add the assertion that Tweety is a penguin:

```
penguin(tweety).
```

The usual process of applying the penguinsDontFly default starts. The logic first determines that there is no more preferred default that would block the application of this default and asserts:

```
defcon(penguinsDontFly(tweety), not(flies(tweety))).
```

The next task is to delete the default applications that are less preferred than this. This deletes the consequences of the application of the default that birds fly so that the following formulas are no longer in the KB:

```
applied_default(birdsFly(tweety), flies(tweety)).
add_default_conclusion(birdsFly(tweety), flies(tweety)).
flies(tweety).
```

And the rest of the application of the penguinsDontFly default proceeds normally and we eventually get:

```
not(flies(tweety)).
```

8.4 Future work

There are several ways this logic could be modified to give better results. These modifications all come at the cost of greater complexity and slower performance. Whether these will be of benefit practically is not clear. We might have a logic that covers more theoretical cases that do not appear often (or ever) in normal usage. The question is then whether the added cost of computation is worth these few cases.

8.4.1 FOL completeness

The logic was based on the assumption that the underlying first order system was complete (except for the reaction to inconsistencies) and this is not the case for Alma. To remedy this situation, we would need to change Alma so that it is complete in the forward direction. This will cause harder control problems however. Another possibility is to use Alma in the backward direction for default queries. This will require backward reasoning through defaults which is not done now. Another approach would be to characterize more closely what is lost in not having this completeness and whether these ares are to be handled.

8.4.2 IDs

As we pointed out above, this implementation does not deal with IDs the way the logic requires and it would seem that there are cases where the implementation would deviate from the logic in the results of the computation. Therefore one of the tasks is to make IDs explicit and to reason about them in the way the logic requires.

8.4.3 Control

The control problem is a general one and applies not only to the mistake handling with non-monotonic language but to Alma in as well and to the logical approach to reasoning in general. In the context of this implementation though, one could search for control heuristics that would make computing defaults more efficient and make the inferences more relevant to the queries of interest.

8.4.4 Inferring preferences

Our nonmonotonic logic requires that the preferences in the domain be explicitly represented. However, in many cases, preferences can be inferred from the structure of the domain. This is particularly the case for hierarchical information where specificity can be used to deduce default preference. This sort of computation can be implemented in the logic using meta-representations of Alma.

8.5 Conclusion

We have presented an implementation of the rules for the previous chapter for handling mistakes using a non-monotonic logic language. The implementation simplified the algorithms and are therefore less general than the version in the previous chapter.

The resulting system was used to encode and test various samples of non-monotonic reasoning found in the literature. For most of the cases, the logics did get the expected answers, in some cases after getting a wrong answer first. This is a characteristic of the system and illustrates the improvement of the answers over time. Tests of the system on more extensive problems are presented in the next chapter.

Chapter 9

Tests for non-monotonic logics

This chapter presents two separate but closely related topics. One is a test suite for non-monotonic logics, the other is the performance of our logic on this test suite.

We have gathered a wide range of problems involving non-monotonic reasoning in the literature and have classified them into categories based mainly on the structure of the problems. Each category has a number of problems with different degrees of complexity. This test suite includes Lifschitz's benchmark problems [104] and also others found in the literature. We present one problem for each category here as illustration. The complete set of problems is found in appendix 11.

The second purpose of this chapter is to give an account of the performance of our logic on that test suite. Some of the categories were not solvable by our logic while others were partially solvable and in others yet, our logic solved all the problems. In this chapter we provide a detailed account of the solution of one problem in each category. The encoding for the other problems in that category that we solved is found in appendix **??**.

9.1 Comparing reasoners vs comparing logics

We begin by justifying the need for a test suite for non-monotonic reasoners. After all, it would seem that the appropriate way to test a reasoner would be to specify it accurately, prove properties about it and compare these properties among different reasoners. The distinction we make here is that we want to compare implemented logics. These may not have as clean a characterization as we may like, and the set of properties might not be easily derived. Also, the dynamic characteristics of the logics need to be taken into account in the comparisons. It is therefore valuable to have a way to test these implementations and our approach has been to gather an array of sample tests from the literature and classify them into categories that seem to require similar capabilities of the reasoner.

9.1.1 KLM

Kraus et al (KLM)[94] characterized consequence relations for non-monotonic logics using a few properties that are believed to be important and useful for non-monotonic reasoners. This work enabled a large variety of logics to be compared according to whether they had or did not have these properties. This was very valuable in that the earlier method of comparing logics essentially consisted in finding examples that one logic could express and reason intuitively with and another couldn't. The new approach made the comparison more systematic.

When it comes to comparing implementations of reasoners though, this approach is not directly applicable since it requires us to first of all derive the KLM properties of the implementation. This may not always be possible for more practically oriented implementations. Things do not always proceed cleanly from a well specified semantics and proof theory to algorithms and to implementations. The non-computability of non-monotonic logics also complicates issues since a logic that will theoretically give us the right answer some day is not as interesting as an implementation that does give us the right answer in a short time. A complement to this approach then is to have a suite of tests that the implementations can be run on and the implementations compared based on the tests that they successfully pass.

9.1.2 Comparing implementations

We have started work on such a suite of tests to test the capabilities of the logic described in the previous chapter. We have gathered a number of examples of non-monotonic reasoning in the literature and have classified them in various categories. The intention is to compare implemented logics through the identity and number of the categories for which they give reasonable answers in a reasonable time span. There are still issues of speed of reasoning, scaling up, ease of encoding and more that we have not addressed. However we think that this is a basis on which to build a better tool for testing implementations of non-monotonic logics.

Ideally, a non-monotonic reasoner should be able to solve all the problems in the test suite. However, that need not be the case. Just as we propose a range of mistake-handling approaches, it is advantageous to have a range of non-monotonic logics which solve different subsets of the test suites. The gain then can be better efficiency or better temporal characteristics in the special cases. However, knowing what cases a particular reasoner can or cannot handle is important.

9.2 Human non-monotonic reasoning

While it is desirable to have non-monotonic reasoners that solve a very wide range of non-monotonic reasoning problems, it is interesting to verify whether people do those sorts of non-monotonic reasoning. After all, a motivation for non-monotonic reasoning has been the sort of reasoning people do everyday as opposed to the more strict reasoning in mathematics and logic.

9.2.1 Everyday reasoning

Galotti [60] surveys the psychological literature on different types of reasoning up to 1989. She makes a first distinction between thinking and reasoning: thinking includes reasoning, problem solving, decision making and brainstorming. Reasoning is taken to be a type of thinking and is defined as a mental activity that transforms information to reach some conclusion. The mental activity has to be focused on some goal and must be consistent with logic if all the premises of the reasoning are specified. However the conclusions need not be deductively valid in general. A further distinction is made between formal and everyday reasoning. Figure 9.1 contrasts the characteristics of these forms of reasoning tasks.

We note that nonmonotonic reasoning problems share some of the characteristics of everyday reasoning but the formalism we develop here seems more apt for tasks that satisfy more of the conditions.

- While tasks given to standard nonmonotonic formalisms might have several answers, the formalisms do not in general have a notion of the quality of the answers. Our formalism will have a notion of the quality of answers: the longer we compute, the better the quality the answer will be.
- Similarly, in our formalism, it is not known when we have reached the 'right' answer. All we can say is that if we compute more, we will not get a worse answer. Therefore it is better suited for tasks where there is not

Formal reasoning	Everyday reasoning
All premises are supplied	Some premises are implicit or not supplied
Problems are self-contained	Problems are not self-contained
There is typically one correct answer	There are typically several correct answers that differ in quality
There usually exist established methods of inference that apply to the problem	There rarely exist established procedures to solve the problem
It is typically unambiguous when the problem is solved	One often wonders if the 'best' solution is good enough
The content of the problem is often of limited academic interest	The content of the problem has personal relevance
Problems are solved for their own sake	Problems are solved as a means of achieving other goals

Figure 9.1: Formal and everyday reasoning

necessarily a point at which one has obtained the correct answer and the computation can stop.

- Finally, we plan to use our formalism in non-toy domains where the tasks are not just of academic interest.

The views of the relationship between formal and everyday reasoning cover the range from formal reasoning being a part of everyday reasoning to the view that they share different processes and share few similarities. An interesting difference noted by [140] to support that last point is that formal reasoning has a "long-chain structure" whereas everyday reasoning has a "fork" structure. The long-chain structure uses many individual steps each leading to the next, whereas in the fork structure there are many short lines of argument, each with some degree of certainty, but which all converge on the solution.

Chapman [29] argues that everyday reasoning includes social and communicative argumentation and develops from argumentation in discourse. Since he claims that everyday reasoning has discourse origins, conversational norms, in particular, Grice's maxims play an important part in everyday reasoning. He explains some of the errors people make in formal reasoning as being caused by the assumption of Grice's maxims in these cases. For example, if told "If it rains the grass is wet" and "The grass is wet", a common error people make is to think that it has rained. This is explained by them applying Grice's maxim of relation and quantity and therefore assuming that no other information is relevant to the situation (for example that the sprinkler is on).

Chapman links everyday reasoning with nonmonotonic reasoning through Grice's maxims which have a "deep kinship" with nonmonotonic reasoning (in chapter **??** we present work that formalizes some of Grice's maxims in a nonmonotonic logic) . In both nonmonotonic reasoning and reasoning with Grice's maxims, inferences are made from an absence of information and the inferences made may have to be later retracted. Logics of conversation and everyday reasoning may have a common structure in which Chapman suggests, nonmonotonic reasoning could be important.

Metareasoning is closely linked to commonsense reasoning in that one could view the jumping to conclusions that nonmonotonic reasoning does as a form of reasoning about what assumptions or axioms the logic should work with and finding the consequences of these. Reiter, for example, takes this view and sees his default rules as meta-rules of inference. Given the similarities between everyday reasoning and commonsense reasoning, we would expect there to be an analogous connection in psychology between everyday reasoning and metareasoning [131, 132]. This does not seem to be the case, and it would be interesting to find the reasons for that.

9.2.2 Human experiments

We now turn to some experiments done on humans to test whether they do default reasoning as is usually taken to be the case.

Elio and Pelletier experiments

The work of Elio and Pelletier [137, 49, 50, 138] focuses on human performance in default reasoning. They argue that default reasoning is "psychologistic" in that there is no notion of correct inference other than what people do. They reject using intuitions to construct formalisms for default reasoning and insist that empirical investigations of human behavior is necessary.

The experiments they did are base on the first four of Lifschitz's benchmark problems. The problems given to the subjects were modified and elaborated with extra information. In these problems there are two objects and a number of default rules. One of the objects is known to be exceptional by not obeying some of the defaults. The problem is to decide whether the other object does obey the defaults.

The results were that people did generally apply the defaults to the object in question but that was influenced by two factors:

- the specificity of the information about the exceptional object and why it violated the rules
- the similarity between the exceptional object and the object people were meant to reason about.

The conjecture of the authors was that people reject the defaults if they can construct alternate scenarios to explain the violations. Other findings are that

- People do better with natural kinds than with artificial kinds–people seem to have less confidence that artificial classes behave like natural classes when it comes to defaults.
- There seems to be a heuristic that if an object is exceptional in one sense, it is likely to be exceptional in other senses also.

These results are not the ones predicted by AI theories where the fact that one object violates one default does not affect its obeying another and where one object violating a default does not affect another object obeying that default. Also there is no distinction made between natural and artificial kinds.

Ford and Billington

[58] gives an account of experiments on default reasoning to investigate people's performance on defeasible and non-defeasible rules. In these experiments they controlled for prior knowledge, beliefs and opinions. The Elio and Pelletier experiments involved elaborate stories that appealed to people's prior knowledge. These experiments, however, was about supposed plants and animals from a far off galaxy so that they were not meaningless but not as loaded with meaning as Elio and Pelletier.

The problem set consisted of 14 problems that systematically varied the interaction between defeasible and non-defeasible information and the degree to which they conflicted. Most AI formalisms they used agreed on all of the problems except for one problem.

In those problems with no conflicts, the results followed the AI predictions very closely. However, when it came to conflicts, the match between human performance and the AI conclusions differed much more. There was an especially large difference when it came to reasoning about conflict with specificity.

Ford and Billington identified some negative and some positive factors from these experiments. Some of the negative factors are that:

- people are reluctant to draw tentative conclusions when faced with conflicting defeasible rules.

- people seem to count the number of paths leading to a conclusion.
- people seem to take the length of the path into account without consideration for the rules along the path

Some positive factors are that:

- people recognize that if all Xs are Ys, then there can be Ys that are not Xs.
- people recognize that if all Ys are Zs then any Xs that are Ys are Zs
- people recognize that if Xs are usually Ys, there are many Ys that are not Xs potentially.

The conclusions Ford and Billington draw are that:

- People don't use specificity.
- People are unwilling to come to conclusions when they have no previous experience in the domain.
- When people do draw conclusions, they are neither coherent nor rational.

The significance of these results

While we agree with Elio and Pelletier on the inadequacy of using intuition for determining the behavior of non-monotonic logics, their emphasis on empirical investigations as the only way to validate non-monotonic reasoning seems misplaced. As they themselves recognize we do not reject first order logic given the results of Wason [188]. What we need then is a justification of default reasoning other than intuition and human performance. One suggestion is that one can consider the utility to an agent operating in the world to have and to use defaults and preferences among defaults. The intuition is that a rational agent would use defaults, it is unclear though what behavior would be optimal. Refusing to use defaults would most certainly paralyze the agent. Using defaults without care for their applicability does not seem to be right either. We believe that some sort of use of defaults with a readiness to correct mistakes might be a good behavior. This sort of study has been done in the context of probabilistic reasoning [167, 84].

The finding that people seem to take more than just the logical structure of the problem into account and the conjecture by Elio and Pelletier that people construct scenarios to explain the violations of the rules and apply these to other objects is very interesting. Logics in general, lack the context sensitivity for these sorts of behavior. This could be realized by an agent who notices an inconsistency or incoherence and instead of simply looking for a default that can fail, it does a more general diagnosis and searches for a explanation which it then uses in further reasoning. This is beyond the scope of the current work but could be explored using Alma as framework.

While we may not be directly interested in modeling human behavior but instead are interested in getting agents to operate effectively in the world, experiments on how humans reason are invaluable since the world the agents operate in are usually designed and invariably described by people and because the agent typically has to interact with people and has to take their actions into account.

9.3 Categories

There are a great many examples of non-monotonic reasoning problems in the literature. The examples seem to fall into those illustrating prototypical behaviors of non-monotonic reasoning and other examples that are meant to show some shortcoming in some formalism. To have a better handle on these, we classified the examples into 12 categories. The criteria for our classification involved the structure of the problem and the sorts of resources required to solve them. There are also several examples that can be mapped onto one another with a simple change of names. In those cases, we picked one of the examples only so that as far as possible, the examples exercise different capabilities of the logics.

9.3.1 The Lifschitz benchmarks

In a well known set of problems, Lifchitz [104] lists a number of benchmark non-monotonic reasoning problems according to the sorts of reasoning they represent. The aim of this benchmark was to verify whether a logic could formalize the problems rather than to implement them. We are however also interested in solutions to these problems in an implementation. Lifschitz has five major classes: default reasoning, inheritance, uniqueness of names, reasoning about action and autoepistemic reasoning. Within each class, he presents several problems that differ in complexity. For example, in the default reasoning class, the examples go from "basic default reasoning" where the main task is a default application, to "reasoning about priorities" where the reasoner is meant to derive a conditional whose antecedent is a statement about the priorities of defaults. While it seems that every reasoner should solve the first example, the last one seems to require more expensive capabilities, including meta-reasoning and the ability to make assumptions and reason with them. Similarly, in the "reasoning about actions" category, the problems range from "frame problem for temporal projection" in which the reasoner has to conclude the results of an action to "counterfactual reasoning about unexpected change".

That classification while appropriate for describing the sorts of problems one may express with non-monotonic logics does not seem well suited for testing the capabilities of a reasoner. The problems within each category seem to require a reasoner with a wide range of capabilities to solve them.

9.3.2 A capability-based classification

We preferred to classify the problems we found, including those from [104] into categories that seem to require similar capabilities. This allows us to possibly compare different reasoners based on the categories of problems that they can solve. For instance, we may say that some reasoner solves the single default cases, but not the multiple inconsistent defaults cases. This classification may give a better measure of the capabilities of implemented non-monotonic reasoners. This classification was based on our reasoner and different reasoners may have different natural classifications. This is to be investigated further.

The categories we picked are: 1. Simple default application; 2. Multiple consistent defaults; 3. Multiple inconsistent defaults with explicit or implicit preferences; 4. Multiple inconsistent defaults with no preference; 5. the Closed World Assumption; 6. Epistemic reasoning; 7. Reasoning about names; 8. Reasoning about action; 9. Reasoning with assumptions; 10. Diagnostic reasoning; 11. Minimization; and 12. Miscellaneous problems.

An account of these categories with examples and our solutions to the examples is presented next.

9.3.3 Reasoning in time

The categories above and indeed all of the examples in the literature do not typically consider the change in the beliefs of the agent doing the reasoning as time passes. This is the case even for cases of temporal reasoning because there the agent is typically seen reasoning about time, as though time is just another variable, rather than reasoning in time–reasoning as the world changes. It seems reasonable to suppose that most agents operating in the world will not have all the relevant information to the solution of a problem presented to them at one time at the start of the reasoning. Rather, information will come a little at a time with the agent possibly concluding false facts during the course of reasoning. This time-sensitivity is a distinctive aspect of active logics and a crucial aspect of any agent that is to operate in the world. Most of these examples found in the literature can be put in a time-situated framework where the relevant information is provided over time rather than all at once and in different orders. The relevant measure then would be the way the KB of the reasoner changes over time rather than its final state.

9.4 Tests

We now describe the categories into which we gathered the non-monotonic reasoning examples that we found and give a representative example from the literature. We show the solution to that example in our logic if available. The encodings are shown as the sentences input into Alma to describe and solve the problem. In our encodings, we do not include the Alma formulas that correspond to the rules for applying defaults or resolving contradictions. The behavior of the system is shown as a sequence of interesting events in the computation and the steps at which they occurred. These interesting events are the addition of interesting formulas by the logic, as well as their distrust and deletion. The addition of formulas to the KB by the user is also included in the list.

9.4.1 Result summary

The results of applying the problems to our logic are summarized below:

Problems solved appropriately	46
Problems not attempted†	23
Problems not solvable‡	24
Total problems	93

† These problems were very similar to problems that had been successfully solved and we did not attempt to solve them. It is expected that they too would be sucessfully solved.

‡ These were problems that could not be expressed in the language or cases where there are no procedures in the language to solve the problems. These include:

- Assumptions. In these problems, the solution required the logic to make some assumption. The logic cannot currently make assumptions and reason about their consequences. This is a deficiency in Alma/Carne and once this is solved, these problems should be solvable.
- Minimization of instances. While we can successfully minimize subsets of large classes (see below for examples), we have not implemented the procedures or the logic to enable this in our logic. Therefore, we cannot minimize the extent of predicates so that we account for only some individuals.
- Diagnosis. These problems involved diagnostic reasoning. The kind of reasoning done in our logic to diagnose contradictions is very simple and cannot be used to solve these problems. This would require a separate diagnostic reasoning module to be added to the logic. This can be modeled logically and make use of the meta-representations available in Alma.

9.4.2 Simple default application

This category of examples consists of applying a default given that the premises are true.

An example

This is example A1 of the benchmark problems [104].

Assumptions:

- Blocks A and B are heavy.
- Heavy blocks are normally located on the table.
- A is not on the table.

Conclusions

- B is on the table.

This illustrates that even though heavy blocks are usually on the table, we do not always conclude that. In this case we know non-defeasibly that A is not on the table, so the default does not apply in this case. In addition, the non-application of the default to A does not affect its application to B.

Our encoding

This is a rather straightforward application of a default.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Blocks a and b are heavy.

and(block(a), heavy(a)).
and(block(b), heavy(b)).

% Heavy blocks are normally located on the table

named(fif(and(block(X), heavy(X)),
  conclusion(defcon(heavyOnTable(X), onTable(X)))),
      heavyOnTable(X)).
default(heavyOnTable(X)).

% a is not on the table
not(onTable(a)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Behavior

Step	Formulas
4	`onTable(b)`

This formula stays in the KB thereafter.

9.4.3 Multiple Consistent defaults

Instead of just one default as above, we now have multiple defaults. These are not inconsistent but may have interesting interactions among one another. Reiter and Criscuolo [161] list a set of examples in all of which the following defaults are present: 1. Typically As are Bs and 2. Typically Bs are Cs. Each example then adds an additional link between these defaults, for example that “No As are Cs”, “As are typically not Cs” and so on.

Example

The example we choose to illustrate this category is one of those from [161]:

1. *Typically As are Bs*
2. *Typically Bs are Cs*
3. *It is not the case that typically As are Cs*

Given an object that is an A, is it a C?

Our encoding

We want to deny that As are typically Cs, which is different from asserting that As are typically not Cs (see [126]). The difference is apparent if we have an object that is an A but not a B. In the first case, there should be no conclusion whereas in the second case we assert that the object is not a C.

The denial is represented by a default that defeats itself: if an object is an A and a C, then it is typically not a C. The consequent of the default causes a contradiction which results in suspending belief about the object being a C. Since the antecedent of this denying default will only be derivable when the fact that the object is a C is derived, we do not get the problem of asserting spurious negations as above. The representation of the denial that As are Cs is best seen as a constraint in the KB–there are typically no cases where something is an A and a C. If there is such a case, then C cannot hold. At this point, the active logic detects the contradiction and removes it which leaves us with no opinion as to whether the object is a C. Note that if we assert that some object is both an A and a C, that will not generate a contradiction because the denial default will not apply. We can take this kind of sentence as expressing denials in our logic.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Typically As are Bs
% Typically Bs are Cs

named(fif(a(X), conclusion(defcon(asbs(X), b(X)))), asbs(X)).
default(asbs(X)).

named(fif(b(X), conclusion(defcon(bscs(X), c(X)))), bscs(X)).
default(bscs(X)).

% It is not the case that typically As are Cs
% This is rendered as If both Ax and Cx, then Cx is not the case.

named(fif(and(a(X), c(X)), conclusion(defcon(denyAC(X), not(c(X))))),
      denyAC(X)).
default(denyAC(X)).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Behavior

Step	Event	
4	Add `a(anna)`	We add this formula
8	Add `b(anna)`	From the default As are Bs
12	Add `c(anna)`	From the default Bs are Cs
16	Add `not(c(anna))`	From the denial default
17	Distrusted `c(anna)`	From the contradiction
17	Distrusted `not(c(anna))`	From the contradiction
17	Added `contra(43, 35, 16)`	From the contradiction

This contradiction does not get resolved because there is no preference for one or the other of the contradictands, so we do not assert anything about `c(anna)`. This seems to be the appropriate response to this problem.

9.4.4 Multiple inconsistent defaults with explicit preferences

In these cases, there are several defaults that could apply because their premises are true, but they cannot apply jointly because that will cause an inconsistency. However we do have explicit preferences between the defaults which helps decide what the contents of the KB should be.

Example

This example is from [5]. The preferences between the defaults are represented by an epistemic entrenchment ordering [69].

1. Sicilians are normally hotheaded.
2. Blondes are normally not hotheaded.

The entrenchment is given by:

$$sicilian(X) \rightarrow \neg hot(X) < sicilian(X) \rightarrow hot(X)$$

$$blonde(X) \rightarrow hot(X) < blonde(X) \rightarrow \neg hot(X)$$

$$sicilian(X) \wedge blonde(X) \rightarrow \neg hot(X) < sicilian(X) \wedge blonde(X) \rightarrow hot(X)$$

Queries:

1. Fiora is Sicilian. Is she hotheaded?
2. Johanna is blonde and German. Is she hotheaded?
3. Rachel is a blonde Sicilian. Is she hotheaded?

Encoding

We represent

$$sicilian(X) \rightarrow \neg hot(X) < sicilian(X) \rightarrow hot(X)$$

This is represented as a two defaults and a preference. A default that Sicilians are typically hotheaded and one that Sicilians are typically not hotheaded. The preference is that the first default is preferred to the second. The facts about blondes are similarly represented.

The last set of formulas:

$$sicilian(X) \wedge blonde(X) \rightarrow \neg hot(X) < sicilian(X) \wedge blonde(X) \rightarrow hot(X)$$

is represented as a preference for Sicilians being hotheaded to blondes not being so. This is not the only way these defaults could be encoded. This approach seems closer to the original expression of the problem.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Typically sicilians are hotheaded
named(fif(sicilian(X),
  conclusion(defcon(sicilianNotHot(X), not(hot(X))))),
      sicilianNotHot(X)).
default(sicilianNotHot(X)).

% Typically sicilians are not hotheaded
named(fif(sicilian(X),
  conclusion(defcon(sicilianHot(X), hot(X)))),
      sicilianHot(X)).
default(sicilianHot(X)).

prefer(sicilianHot(X), sicilianNotHot(X)).

% Typically blondes are hotheaded
named(fif(blonde(X),
  conclusion(defcon(blondeNotHot(X), not(hot(X))))),
      blondeNotHot(X)).
default(blondeNotHot(X)).

% Typically blondes are hotheaded
named(fif(blonde(X),
  conclusion(defcon(blondeHot(X), hot(X)))),
      blondeHot(X)).
default(blondeHot(X)).

prefer(blondeNotHot(X), blondeHot(X)).

% This takes the place of the third set of formulas
prefer(sicilianHot(X), blondeNotHot(X)).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Behavior

Step	Event	
5	Add `sicilian(fiora)`	We add this to the KB
9	Add `hot(fiora)`	Since Fiora is Sicilian, she is hotheaded
12	Add `german(johanna)`	We add this to the KB
12	Add `blonde(johanna)`	We add this to the KB
16	Add `not(hot(johanna))`	This is from Johanna being blonde
28	Add `blonde(rachel)`	We first just add that Rachel is blonde
32	Add `not(hot(rachel))`	This is because Rachel is blonde
38	Add `sicilian(rachel)`	We later find that Rachel is Sicilian
40	Delete `not(hot(rachel))`	Since Rachel is Sicilian, the blondeNotHot default is not applicable
42	Add `hot(rachel)`	The default that Sicilians are hotheaded wins

Note that we add the information about Rachel at two different times. We first state that Rachel is blonde which lead to the conclusion that she is not hotheaded. When we later add the fact that she is Sicilian, the previous conclusion cannot stand and it is removed from the KB before adding the conclusion that she is hotheaded. This illustrates the "reasoning-in-time" aspect of active logic.

9.4.5 Multiple inconsistent defaults with implicit preferences

In this case, there are inconsistent defaults and the examples do not specify any preference between them. However, by inspecting the problem, it is clear that there are implicit preferences. These are typically specificity preferences: if an object a of class C has property P and an object b is in class D which is a subclass of C and has property $\neg P$, then the apparent contradiction between these properties is resolved by picking the property associated to the more specific class. In this case, we take b to be $\neg P$.

Our logic does not derive these specificity preferences so these problems cannot be solved appropriately unless we make explicit the preferences. Having the logic infer the preferences can be done using the meta-representations possible in Alma. See the examples on minimization for how this could be done.

We do not present any example of this here since if we add preferences explicitly, this becomes like the above case, and if we don't, it becomes like the next case. The appendix, however, has examples of this kind of problem.

9.4.6 Multiple inconsistent defaults with no preferences

In this case, there are defaults that lead to inconsistency but no preferences, either explicit or implicit so the logic is given no indication as to which default it should prefer. Our logic finds IDs and ends up with no opinions about the facts of interest although it might initially assert that the facts are true or false. In this case too, the logic behaves as expected.

Example

The best known example of this type is the Nixon Diamond problem [161].

Typically republicans are not pacifists.
Typically quakers are pacifists.
John is both a quaker and a republican. Is John a pacifist?

Encoding

The defaults are encoded in the usual way, without any preference between them:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Republicans are typically not pacifists
named(fif(republican(X),
  conclusion(defcon(repnotpac(X), not(pacifist(X))))),
      repnotpac(X)).
default(repnotpac(X)).

% Quakers are usually pacifist
named(fif(quaker(X),
  conclusion(defcon(quakerpac(X), pacifist(X)))),
      quakerpac(X)).
default(quakerpac(X)).

% John is republican and quaker
and(republican(john), quaker(john)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Behavior

In this case, there are no conclusions about John.

9.4.7 Closed World Assumption

The closed world assumption is made by assuming facts are false unless we know otherwise. It is not possible and not practical if it were possible, to generate the negation of all formulas in our logic. Instead, when using the CWA, we need to make its use explicit in the formulas it is needed in. This is a shortcoming of the forward chaining approach of Alma.

The CWA is represented as a default that expresses that the formula we are interested in is false, and that default is less preferred than the other defaults. So the logic usually derives the formula is false but any evidence that it is not so causes the logic to change its KB.

Example

This example is from [156]:

Consider an extensional database with

$$Teacher = \{a, b, c, d\}$$
$$Student = \{A, B, C\}$$

and the Teach relation: $(a, A), (b, B), (c, C), (a, B)$.

Query: who does not teach B?

The answer we want is $\{c, d\}$.

Encoding

Query answering is not very elegant in active logic because of the forward chaining nature of the logic. One solution is to start a backward search for the answer. Another is to allow the logic to derive the answer in its usual forward chaining mode. This is the approach we take here.

An axiom is used to generate queries about whether each teacher teaches the student. The default implementing the CWA concludes that each of the queries is false. However, if there is a fact that shows the query to be true, the default does not apply. And if the default does apply, denying the query, and later on, there is a derivation that the query is true, the denial is removed.

Note that this solution does not wait for proof that the query cannot succeed as would be expected for the usual implementations of the CWA. It assumes the query is false, and if we have better information later, it changes its mind.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Any query is assumed false.
named(fif(query(X), conclusion(defcon(cwa(X), not(X)))), cwa(X)).
default(cwa(X)).

% Facts about teachers and students
teacher(tA).
teacher(tB).
teacher(tC).
teacher(tD).

student(sA).
student(sB).
student(sC).

teaches(tA, sA).
teaches(tB, sB).
teaches(tC, sC).
teaches(tA, sB).

% For each teacher, we query wheter that teacher teaches sB
fif(teacher(X), conclusion(query(teaches(X, sB)))).
```

Behavior

For each teacher tX who does not teach B, we get $not(teaches(tX, B))$.

Step	Event	
2	query(teaches(tD, sB))	From the querying axiom
2	query(teaches(tC, sB))	From the querying axiom
2	query(teaches(tB, sB))	From the querying axiom
2	query(teaches(tA, sB))	From the querying axiom
6	not(teaches(tD, sB))	From the default
6	not(teaches(tC, sB))	From the default

This example illustrates a shortcoming of the Alma system as far as queries are concerned. Since the system is essentially forward chaining, the query had to be transformed to be in the forward direction. The backward chaining facility in Alma does not assume the CWA or negation-as-failure as in Prolog. Therefore if that were used, the negation would not have been obtained directly.

9.4.8 Epistemic reasoning

Epistemic reasoning or auto-epistemic reasoning is another popular domain for non-monotonic reasoning in the literature. In our implementations, we interpreted these problems in two ways. If the problem used epistemic reasoning to solve other problems, we translated these problems to use the default reasoning formalism of the logic which does not explicitly use any auto-epistemic information. For those examples that are meant to illustrate epistemic reasoning, the introspection feature of Alma was used.

Example

This is example E4 from [104]

Assumptions:

- Blocks that are not known to be heavy are on the table.
- Block A is heavy.

Conclusions

- Block B is on the table.

Encoding

We use the negative introspection ability of Alma to represent the rule about blocks being on the table. We also need to assert that B is a block.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% a is a heavy block
and(block(a), heavy(a)).

% b is a block
block(b).

% blocks that are not known to be heavy are on the table.
if(and(block(X), not(eval_bound(pos_int(heavy(X)), [X]))),
   onTable(X)).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Behavior

Step	Event	
3	onTable(b)	The rule is applied and B is on the table.

9.4.9 Reasoning about names

The issue in reasoning about names is to reason about the uniqueness of names. The examples we found were straightforward cases of default application except that one had to make the distinction between objects and names of objects. This was implemented in the logic by using a predicate `names` that maps names to objects.

Example

We illustrate this with example C1 from [104]

Assumptions:

- Different names normally denote different objects
- The names “Ray” and “Reiter” denote the same person.
- The names “Drew” and “McDermott” denote the same person.

Conclusions

- The names “Ray” and “Drew” denote different people.

Encoding

We use a predicate “names” to represent the names of objects in the domain and we have a default that if two objects have different names then they are different. However, “Ray” and “Reiter” name the same person as do “Drew” and “McDermott”.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
named(fif(and(names(X1, Y1),
      and(names(X2, Y2),
  eval_bound(\+ X1 = X2, [X1, X2, Y1, Y2]))),
  conclusion(defcon(dndo(X1, X2), different(Y1, Y2)))),
      dndo(X1, X2)).
default(dndo(X1, X2)).

% ray and reiter denote the same thing
fif(and(names(ray, X), names(reiter, Y)),
    conclusion(same(X, Y))).

% drew and mcdermott denote the same thing
fif(and(names(drew, X), names(mcdermott, Y)),
    conclusion(same(X, Y))).

% same is not different
if(same(X, Y), not(different(X, Y))).

names(ray, p1).
names(drew, p2).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Behavior

step	Event	
4	Add `different(p2, p1)`	p1 and p2 are different by default
4	Add `different(p2, p2)`	p1 and p2 are different by default

9.4.10 Reasoning about action

Reasoning about action is an important class of non-monotonic reasoning applications. Some of the earliest motivation for non-monotonic reasoning was reasoning about action [109]. There are several such examples in the literature, represented in different formalisms, mainly situation based [109, 159] or interval based[2, 92, 93]. Both of these ways of reasoning about action can be represented in the logic. The same non-monotonic behavior is used in both sorts of examples. The logic gives the results expected.

Example 1

This example is D1 from the benchmark problems [104]. Lifschitz uses a situation calculus approach to the representation. Actions are represented as defaults with the preconditions and the fact that the action is executed as the antecedent of the default and the result of the action as consequent. The law of inertia is also represented as a default that maintains all properties by default while any action is done. The defaults representing actions are stated to be preferred to inertia for the facts that they change only. This results in an action changing only the facts specified and leaving all others unchanged.

Assumptions:

- After an action is performed things normally remain as they are.
- Any time a robot grasps a block, the block will be in the hand.
- If a block is in the hand then, after the robot moves it to the table, the block will be on the table.
- Initially block A is not in the hand.
- Initially block A is not on the table.

Conclusions

- After the robot grasps block A, waits, and then moves it onto the table, the block will be on the table.

Encoding

This is a straightforward example. As each action is performed, the defaults enable the correct state of the world to be computed. The action defaults have to be explicitly preferred to the inertia rule applied to the same action.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Inertia: after an action is performed things normally stay the same
named(fif(and(do(A, S), holds(F, S)),
  conclusion(defcon(inert(A, F, S), holds(F, done(A, S))))),
      inert(A, F, S)).
```

```
default(inert(A, F, S)).


% any time the robot grasps a block it will be in its hand
named(fif(do(grasp(B), S),
  conclusion(defcon(graspinhand(B, S),
    holds(inhand(B), done(grasp(B), S))))),
      graspinhand(B, S)).
default(graspinhand(B, S)).

% this is preferred over inertia that is about a grasp action
prefer(graspinhand(B, S), inert(grasp(B), _, S)).

% if a block is inhand, after moving it to the table, it will be on the table
named(fif(and(holds(inhand(B), S), do(move(B), S)),
  conclusion(defcon(moveontable(B, S),
    holds(ontable(B), done(move(B), S))))),
      moveontable(B, S)).
default(moveontable(B, S)).

% This is preferred to inertia.
prefer(moveontable(B, S), inert(move(B), inhand(B), S)).

named(fif(and(holds(inhand(B), S), do(move(B), S)),
  conclusion(defcon(movenohand(B, S),
    not(holds(inhand(B), done(move(B), S)))))),
      movenohand(B, S)).
default(movenohand(B, S)).
prefer(movenohand(B, S), inert(move(B), inhand(B), S)).

% initially block a is not in hand
not(holds(inhand(a), s0)).

% initially block a is not on table
not(holds(ontable(a), s0)).

% robot grasps, waits, and moves
do(grasp(a), s0).
do(wait, done(grasp(a), s0)).
do(move(a), done(wait, done(grasp(a), s0))).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Behavior

Step	Event	
2	Add `holds(inhand(a), done(grasp(a), s0))`	Grasp action
5	Add `holds(inhand(a), done(wait, done(grasp(a), s0)))`	Inertia
6	Add `not(holds(inhand(a), done(move(a), done(wait, done(grasp(a), s0)))))`	Move
6	Add `holds(ontable(a), done(move(a), done(wait, done(grasp(a), s0))))`	Move

Note that inertia about the location of the block does not come into play when the agent grasps or moves the block.

Approach 2

In this example from [168], we do not use a situation-based representation since we have to reason about intervals.

Facts are taken to be true at times rather than in situations. This causes a difficulty for the general approach we have here. We need to progress the whole world through each time interval. This is done through an $applyInertia$ action. Progressing the world is easier in the case of situations since the actions that led to the situation are explicit in the representation of the situation itself.

We represent actions using three statements: one representing starting the action, one for the ending and one for the effects of the action. The effects of the action are represented as a default that is preferred to inertia. We assume rather unrealistically that the effects of the action are effected only at the end of doing the action. Inertia fulfills the same role as before, but in this case it is expressed in time intervals.

Example

Consider a box, B, and a car, C, both of which are located in the city of Linkoping at time 0, which represents the beginning of the scenario. The box is not in the car at time 0. Two action types are considered, namely to load a box into the trunk of a car, and to drive a car to a specified city. From time 8:15 to time 8:20 the box is loaded into the car; from time 8:40 to time 11:15 the car is driven to Stockholm. Question: where is the box at time 13:00?

Answer: the box is in Stockholm

Encoding

Note that inertia is now specified between two time intervals and we use a predicate `trueAt` to represent what is true at particular times. This encoding is not the most perspicuous and could be improved. It does however, illustrate how the logic can be used in cases like this.

```
% Inertia
named(fif(and(applyInertia(Ti, Tj), trueAt(X, Ti)),
  conclusion(defcon(inertia(X, Tj), trueAt(X, Tj)))),
      inertia(X, Tj)).
default(inertia(X, Tj)).

%___________________________________________________
%% loading a box
```

```
% starting the load

fif(start(load(X, Y), Ti), conclusion(started(load(X, Y), Ti))).

% ending the load
fif(and(end(load(X, Y), Tj),
       started(load(X, Y), Ti)),
   conclusion(done(load(X, Y), Tj))).
% the effects of the load:
named(fif(done(load(X, Y), Tj),
  conclusion(defcon(loadIn(X, Y, Tj), trueAt(in(X, Y), Tj)))),
      loadIn(X, Y, Tj)).
default(loadIn(X, Y, Tj)).
% we prefer this to inertia
prefer(loadIn(X, Y, Tj), inertia(not(in(X, Y)), Tj)).

%__________________________________________________
%% driving
% starting the drive
fif(and(start(drive(V, X, Y), Ti),
       trueAt(inCity(V, X), Ti)),
   conclusion(started(drive(V, X, Y), Ti))).
% ending the drive
fif(and(end(drive(V, X, Y), Tj),
       started(drive(V, X, Y), Ti)),
   conclusion(done(drive(V, X, Y), Tj))).
% the effects of the drive: we leave the city we started out from
named(fif(done(drive(V, X, Y), Tj),
  conclusion(defcon(driveNotInCity(V, X, Tj),
    trueAt(not(inCity(V, X)), Tj)))),
      driveNotInCity(V, X, Tj)).
default(driveNotInCity(V, X, Tj)).
% we prefer this to inertia
prefer(driveNotInCity(V, X, Tj), inertia(inCity(V, X), Tj)).

if(trueAt(in(O, V), Tj),
   prefer(driveNotInCity(V, X, Tj), inertia(inCity(O, X), Tj))).

% the effects of the drive: we reach the destination
named(fif(done(drive(V, X, Y), Tj),
  conclusion(defcon(driveInCity(V, Y, Tj),
    trueAt(inCity(V, Y), Tj)))),
      driveInCity(V, Y, Tj)).
default(driveInCity(V, X, Tj)).
prefer(driveInCity(V, X, Tj), inertia(not(inCity(V, Y), Tj)).
if(trueAt(in(O, V), Tj),
   prefer(driveInCity(V, X, Tj), inertia(not(inCity(O, Y), Tj)))).

%__________________________________________________
% general rule about containment
```

```
fif(and(trueAt(in(X, Y), Ti),
       trueAt(inCity(Y, C), Ti)),
   conclusion(trueAt(inCity(X, C), Ti))).

fif(and(trueAt(in(X, Y), Ti),
trueAt(not(inCity(Y, C)), Ti)),
    conclusion(trueAt(not(inCity(X, C)), Ti))).

% general rule about trueAt
fif(trueAt(not(X, T)), conclusion(not(trueAt(X, T)))).
%__________________________________________________
% Initial conditions
trueAt(not(in(box, car)), 0).
trueAt(inCity(car, linkoping), 0).
%__________________________________________________
% the sequence of assertions that represents the scenario.
% We can assert them in the correct sequence or all at once. It is
% easier to understand if we assert them in the right order though.
/*
af(applyInertia(0, 815)).
af(start(load(box, car), 815)).
af(applyInertia(815, 820)).
af(end(load(box, car), 820)).
af(applyInertia(820, 840)).
af(start(drive(car, linkoping, stockholm), 840)).
af(applyInertia(840, 1115)).
af(end(drive(car, linkoping, stockholm), 1115)).
af(applyInertia(1115, 1300)).
*/
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Behavior

We assert the actions above one at a time to illustrate how the answer is built.

Step	Event	
4	Add `applyInertia(0, 815)`	Added to the KB
8	Add `trueAt(inCity(car, linkoping), 815)`	Derived by the logic
8	Add `trueAt(not(in(box, car)), 815)`	Derived by the logic
11	Add `start(load(box, car), 815)`	Added to the KB
12	Add `started(load(box, car), 815)`	Derived by the logic
15	Add `applyInertia(815, 820)`	Added to the KB
19	Add `trueAt(inCity(car, linkoping), 820)`	Derived by the logic
19	Add `trueAt(not(in(box, car)), 820)`	Derived by the logic
22	Add `end(load(box, car), 820)`	Added to the KB
23	Add `done(load(box, car), 820)`	Derived by the logic
25	Delete `trueAt(not(in(box, car)), 820)`	Derived by the logic
27	Add `trueAt(in(box, car), 820)`	Derived by the logic
28	Add `trueAt(inCity(box, linkoping), 820)`	Derived by the logic
31	Add `applyInertia(820, 840)`	Added to the KB
35	Add `trueAt(inCity(box, linkoping), 840)`	Derived by the logic
35	Add `trueAt(inCity(car, linkoping), 840)`	Derived by the logic
35	Add `trueAt(in(box, car), 840)`	Derived by the logic
39	Add `start(drive(car, linkoping, stockholm), 840)`	Added to the KB
40	Add `started(drive(car, linkoping, stockholm), 840)`	Derived by the logic
44	Add `applyInertia(840, 1115)`	Added to the KB
48	Add `trueAt(inCity(box, linkoping), 1115)`	Derived by the logic
48	Add `trueAt(inCity(car, linkoping), 1115)`	Derived by the logic
48	Add `trueAt(in(box, car), 1115)`	Derived by the logic
52	Add `end(drive(car, linkoping, stockholm), 1115)`	Added to the KB
53	Add `done(drive(car, linkoping, stockholm), 1115)`	Derived by the logic
55	Delete `trueAt(inCity(box, linkoping), 1115)`	Derived by the logic
55	Delete `trueAt(inCity(car, linkoping), 1115)`	Derived by the logic
57	Add `trueAt(not(inCity(car, linkoping)), 1115)`	Derived by the logic
57	Add `trueAt(inCity(car, stockholm), 1115)`	Derived by the logic
58	Add `trueAt(not(inCity(box, linkoping)), 1115)`	Derived by the logic
58	Add `trueAt(inCity(box, stockholm), 1115)`	Derived by the logic
62	Add `applyInertia(1115, 1300)`	Added to the KB
66	Add `trueAt(inCity(box, stockholm), 1300)`	Derived by the logic
66	Add `trueAt(inCity(car, stockholm), 1300)`	Derived by the logic
66	Add `trueAt(in(box, car), 1300)`	Derived by the logic
66	Add `trueAt(not(inCity(box, linkoping)), 1300)`	Derived by the logic
66	Add `trueAt(not(inCity(car, linkoping)), 1300)`	Derived by the logic

Note that in this case, the logic is being used to reason about the trip. It can also be used as an on-board reasoner for the trip, coming to believe and facts as the trip unfolds.

9.4.11 Reasoning with assumptions

There are a number of examples in the literature (see appendix XXX) where one has to assume a fact and reason based on this assumption. The assumption may not always be explicit but the most convenient solution to the problem seems to require that the assumption be made. Alma does not reason with assumptions and so this class of examples cannot be done in our logic. Alma can be modified to deal with assumptions in the same way it does backward chaining proofs currently and in that case these problems will be solvable. This is to be done in the future.

9.4.12 Example

An example of one such problem is the Hiding Turkey scenario from [168].

The world is as in the Yale Shooting Problem with two additional fluents: deaf-turkey and hiding-turkey. If the turkey is not deaf, then when gun is loaded it goes into hiding. Firing only kills turkey if it is not hiding.

Initially the turkey is alive, not hiding, and the gun is not loaded. It is unknown if turkey is deaf. The events are: load gun; wait; fire.

The conclusion we want to reach is that either the turkey is deaf and dead or non-deaf and alive.

The solution of the problem seems possible if we can assume at the beginning that the turkey is deaf and find the consequences for this, then assume that it is not deaf and find the new consequences. Since the turkey has to be deaf or not-deaf, we can then get the results desired. However, this sort of reasoning is not available in Alma.

9.4.13 Diagnostic reasoning

Several examples of non-monotonic reasoning involve diagnosing problems. Diagnosis using the defaults asserted in the logic has not been explored in our system. Using non-monotonic logic for diagnosis has been studied before.

[158] gives an account of diagnosis from first principles with in a non-monotonic setting. The approach is to use logic to describe the system as a set of interacting components, each of which behaves correctly provided it is not abnormal. Defaults are used to assume that none of the components is abnormal. Observations of the system are asserted as first order sentences. If the observations are of a malfunctioning system, not all the components are normal, therefore some of the defaults cannot be applied. The task then is to find those combinations of defaults that are not used so that the extensions of the system entail the observations.

In our case, the assumption that all components are functioning normally will lead to a contradiction with the observation of a malfunction. The CR rule will revise the assumptions that the components are normal so that we regain consistency. However, the simple ways we have of determining which defaults are not to hold might not be sophisticated enough to determine the best diagnoses.

Another approach to the problem is described in [33]. In this work, non-monotonic reasoning is used to do abduction. This can be used to diagnose problems and malfunctions. The approach is to convert the logical description of the domain using predicate completion and inferring deductively the abductive conclusions based on this transformed theory and the observations. This is similar to the minimization examples we have elsewhere and could be done in our logic with the caveats expressed as regards minimization.

Example

This is example D4 of [104] (Lifschitz). It is ostensibly about actions but there is a diagnostic reasoning/abduction component to it.

Assumptions:

- After an action is performed, things normally remain as they are.
- When the robot grasps a block, the block will normally be in the hand.
- When the robot moves a block onto the table, the block will normally be on the table.
- Moving a block that is not in the hand is an exception to this rule.

- Initially block A was not on the table.
- After the robot moved A to the table and then waited, A was on the table.

Conclusions

- Initially A was in the hand.

To deduce this conclusion, the logic needs to infer what should be the case for the block to be on the table after the moves. We do not want to specify special defaults for that. The logic should make use of the defaults specifying the effects of actions to do that. This would require using the defaults in the backward direction which the rules for default application don't support currently.

9.4.14 Minimization

Minimization problems involve the logic deducing that all members of a class except for the known exceptions have some property. For example, assume that the reasoner knows that birds usually fly and that penguins usually don't fly, and this is all it knows about flying birds. Minimization causes the reasoner to assert that all birds except for penguins fly. Similarly, if the reasoner knows that penguins usually swim but Joe and Fred are penguins that don't, then minimization will lead it to conclude that all penguins except for these two swim.

There are two sorts of examples that deal with minimization of predicates. One kind are those examples where one is interested in computing the truth of some formula and minimization of defaults is simply a way of doing so, as in circumscription. In the second kind, minimization is the objective of the reasoning.

In the first case, we can solve the problems without doing explicit minimization by translating them to our formalism. Instead of circumscribing a formula, we use it in an appropriate default and the same problems can be solved.

In the second case, the logic can be programmed to do the minimization using the meta-reasoning capabilities of Alma. However, it is not clear that this is a behavior one would necessarily want in a general non-monotonic logic since it implies that we already know all the relevant information. Going back to the birds and penguins example, asserting that all birds except for penguins fly seems to be reasonable only if we have some assurance that we know all of the possible counterexamples to the default. This seems to be rarely the case. Indeed, a characteristic of non-monotonicity is uncertainty and we should expect exceptions to any default [51]. Therefore explicit minimization is not a property that we want a non-monotonic logic to have.

Our logic does do minimization in the case of exceptional subsets. It can be programmed to minimize based on individuals in the same way. This last feature has not been implemented yet however.

Example

This is example B1 from [104]

Assumptions:

- Animals normally do not fly.
- Birds are animals.
- Birds normally fly.
- Ostriches are birds.

- Ostriches normally do not fly.

Conclusions

- Animals other than birds do not fly.
- Birds other than ostriches fly.
- Ostriches do not fly.

If the assumptions are *all* that is true about the world, then the conclusions seem reasonable, but that is not the case. Bats are animals that fly and penguins are birds other than ostriches that do not fly.

Encoding

This behavior is not part of the default reasoning rules of our logic. However, Alma can be used to specify this sort of computation. This is what was done here. This required a fair amount of prolog code to manipulate the defaults and other formulas.

```
% animals usually do not fly
named(fif(animal(X), conclusion(defcon(animalsDontFly(X), not(flies(X))))),
      animalsDontFly(X)).
default(animalsDontFly(X)).

% birds are animals
if(bird(X), animal(X)).

% birds normally fly
named(fif(bird(X), conclusion(defcon(birdsfly(X), flies(X)))), birdsfly(X)).
default(birdsfly(X)).

% prefer(birdsfly(X), animalsDontFly(X)).

% ostriches are birds
if(ostrich(X), bird(X)).

% ostriches normally do not fly.
named(fif(ostrich(X), conclusion(defcon(ostrichesdontfly(X), not(flies(X))))),
      ostrichesdontfly(X)).
default(ostrichesdontfly(X)).

fif(and(find_exceptions(X),
and(default(Y),
    eval_bound(is_an_exception(Y, Z, X), [X, Y]))),
    conclusion(exception(Y, Z, X))).

% exception(Y, Z, X) means that the formula named Y says that the cpndition
% Z is an exception to the default named X.
% that is, it is consistent for the antecedent of X and Z to be true but
% in those cases, the default X fails.
```

```
fif(and(find_qualification(D),
eval_bound(qualify_default(D, Q), [D])),
    conclusion(assert_cnf(Q, if))).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

9.4.15 Miscellaneous

There are some miscellaneous problems that do not seem to fit in any of the categories above. These include reasoning with inconsistent justifications, and reasoning about integers.

9.5 Conclusion

We have presented an example from each category of a proposed test suite for non-monotonic logic implementations and have shown how our logic performs in that suite. There is a lot of work that needs to be done on the test suite to improve its usefulness, this being only the first sketch of such a tool. We have also shown that our logic performs as expected in a large variety of examples, however, there still are classes of examples that it has difficulty with. That too is scope for future work. More examples of the problems in the test-suite and more results of out logic are found in the appendices.

Chapter 10

A design for mistake handling in agents

In this chapter we present a design for an agent that reasons and acts in a changing world with incomplete and uncertain information.

The previous chapters were concerned with more limited agents which could be implemented with relatively simple algorithms. We now relax some of the simplifying assumptions used earlier and the algorithms have to be designed accordingly. We need to make more of the representations and procedures explicit and declarative. This makes the system more flexible and capable, but also more complex and less efficient.

The benefits we expect include the ability to handle a large range of mistakes and errors without having to explicitly have a solution for every possible mistake. Especially interesting is the ability to predict that an action or plan that the agent has decided to execute is mistaken before any actions are taken in the world. This however comes at the cost of greater complexity and slower execution. We consider how this can be mitigated by a hybrid procedural/logical system where the logic only intervenes in the operation of the system in case of errors.

Although such a system has not been implemented yet, we conjecture that Alma provides representations and reasoning processes that will facilitate such an implementation. Verifying this is part of future work.

We start by looking at some examples and at how these differ from the kinds of examples we were considering earlier. We next consider what that entails for the representation of the world. We will need more complex representations to map to the more complex structure of the world now. The reasoning processes too need to be changes to make them more capable. We consider those in the following section.

10.1 Examples

We use the following examples to illustrate some of the behavior the logic is designed to model.

1. At 10:00 the agent decides that a file is an obsolete version of its work and so it deletes it. At 10:01 it realizes that the file was in fact the latest version. But now it can't undelete the file.

   In contrast to the mistaken beliefs considered earlier, the actions in this case are not easily reversible. And the solution to the mistake might involve other actions in the world. This has to be specified in the domain specifications.

2. At 10:00 the agent observes that the light is red, so it decides not to cross the road. At 10:02, the agent observes that the light is green and it crosses the road. It was not a mistake not to cross at 10:00 because the agent was

not mistaken in believing that the light was red because the light was indeed red and it changed to green later.

The world changes and the agent is aware that it did, so the earlier decision based on the previous state of the world was not mistaken even though the information then contradicts the current beliefs.

3. At 10:00 the agent observes that an apple is red. At 10:01 it observes that that apple is black, and concludes that it is rotten and decides to throw it away later. At 10:02 it realizes that the ambient light color changed to green at 10:01. So the apple was in fact red and the decision to throw it away is mistaken. Since it has not done so yet, it just removes its intention to throw away the apple.

   In this case the world initially changes without the agent being directly aware of it. The perception it has then conflicts with earlier ones and it decides that the new beliefs are correct and plans to act accordingly. The agent later realizes that there was a change in the world and therefore that the perception was mistaken. It then has to undo the intention. Here we see that beliefs believed to be mistaken might be found not to be mistaken with more information later. The earlier determination of a mistake is itself mistaken.

4. At 10:00 the agent observes the dog in the yard and at 10:01 it lets the cat out in the street believing the dog to still be in the yard. At 10:02 the agent observes the dog in the street. At some point between these two times the dog left the yard and the cat is not safe. The agent does not know whether the dog was in the yard at 10:01 so it may have been a mistake to believe that the dog was in the yard then and it may have been a mistake to let the cat out.

   In this case the agent assumes that the world does not change and takes an action based on this assumption. Later observations imply that that action may have been mistaken. The observations are not sufficient to determine exactly whether they had been mistaken or not but prudence recommends that that possibility be taken into account.

5. At 10:00 the agent learns that Tweety is a bird, so he infers that Tweety flies. At 10:10 the agent observes that Tweety is a penguin, so Tweety does not fly. The agent was mistaken about Tweety's flying as from 10:00.

   The facts in the world in this case do not change. The knowledge of the world that the agent has changes. The agent reasons that this is the case and therefore that its earlier decisions were mistaken.

10.1.1 Differences with the non-monotonic reasoner

We see in these examples that there are interesting interactions between:

1. How facts in the world change.
2. How the beliefs of the agent change.
3. The various relations between the beliefs of the agent and the state of the world.
4. The irreversibility of actions.
5. The knowledge by the agent of how some facts in the world change.

These interactions are not present in a purely logical agent, so the algorithms devised earlier about mistaken beliefs are not likely to be applicable here. There are three main areas of difference between this and the previous reasoner. These concern the causes of the mistake, the effects of the mistake and the fact that the agent is now situated in time.

The agent now reasons about facts that change truth value in time while it is reasoning. This contrasts with the previous case where the facts asserted did not typically change. All the relevant information to the problem was presented at the start of the computation. The agent in that case could see time as just another variable that did not have any relation to its reasoning. This has consequences for the representation and inference in the agent. The fact that the reasoning is time-situated becomes important now.

The causes of mistakes in the previous case were mainly due to a lack of information or of computing resources on the part of the agent. In this case, the world can change, making the determination of mistakes more complex. The agent cannot assume the world is constant and so to determine whether a belief is mistaken, it has to determine what was true in the world at the time it made the decision. Depending on how the world changed, an apparent error may or may not be a mistake. If the change in belief corresponded to a change in the world, there was no mistake. We can no longer automatically associate a contradiction with a mistaken belief.

The effects of the mistake are also different now. In the previous case, the only effects of the inference rules were to add and delete formulas from the KB. These are easily reversible. However, an agent that is acting in the world, will execute actions that are not so easily reversible. The undoing of the results of a mistake need to be more carefully considered since some actions are irreversible.

The result of these factors is that the representation of facts differs from that in the previous cases, and the handling of mistakes is made more explicit and less procedural. There is a need to extend the framework of the previous chapter to address these issues.

10.2 Representation

The possibility that the world might change and the fact that the agent reasons in time lead to the need for a different representation. These representations are useful for agents that deal with mistakes in general. Such an agent has to represent:

- **Beliefs in time** Explicitly representing beliefs allows mistaken beliefs to be detected and represented. If beliefs are not explicit, there is a limited set of responses possible to failures. If the beliefs are represented in such a way that they can be related to the times at which they are held, it becomes possible to detect mistakes in the past. This can be important since mistakes in the past can influence current beliefs, intentions and actions. Otherwise, the agent seems limited to responding to the immediate detection of a mistake if it does not have the ability to track mistakes in the past.
- **Actions, observations, plans, intentions** This is the converse of the above. Without representation of intentions, actions, observations and plans, the agent is limited to working with mistaken beliefs as we have seen earlier.
- **Relation between events and beliefs** The events that the agent observes or produces should be related to its beliefs so that it can keep track of the state of the world and for its actions to be reasoned about.
- **Relation between truth and beliefs** If the agent cannot distinguish between truth (as it perceives it) and beliefs (old ones), it can't detect mistakes. The truth the agent perceives is itself a belief, but at a different time.
- **Expectations** Expectations help the agent discover that its actions did not have the desired effect and detect mistaken actions. If the agent has no expectations of the results of its actions, it becomes harder to detect and identify mistakes.

These representations need not be explicitly present in all agents that handle failures–they can be present to a degree. The flexibility of the agent to deal with mistakes will depend on the form of these representations. The efficiency of the agent is likely to be affected by those factors too.

We present the design of these representations for our agent next.

10.2.1 Representing beliefs in time

A major difference between the problem we address here and the reasoning presented previously (even for the reasoning about action examples) is that in the present case, the agent reasons in time, at the same time as the facts in the world could be changing. In the previous case, the reasoner was outside the changing time and reasoned about it.

The formulas in the KB should therefore represent the changing world. As the world changes, various formulas in the KB become true and false and we would like the KB to reflect these changes. If the light is red now and something like $LightColor(Red)$ is in the KB, when the light changes to green we would like that that formula to be replaced by $LightColor(Green)$. The representation used to represent that the light is now red will affect the computation of the logic.

We consider two approaches: explicitly representing the time at which the formula is taken to be true in the formula itself and not representing the time.

Explicitly representing time

This alternative explicitly represents the time at which a fact is taken to be true in the KB. The facts are taken like eternal facts [152].

Three ways of doing this are:

- Adding an argument representing time to all predicates that are liable to change truth value over time. That argument represents the time at which the predicate is true so that $LightColor(Red, 11:01)$ means that the light is red at 11:01.
- Another way is to use a $Holds$ predicate to represent that a fact holds and to represent facts as terms instead of predicates, for instance $Holds(LightColor(Red), 11:01)$. Here $LightColor$ is not a predicate anymore.
- A third way related to the above is to use a modal operator with the appropriate rules of inference so that $LightColor(Red)@11:01$ means that the light is red at 11:01.

If the light turns green at 11:02, the agent does not make the mistake of believing that it is red at that time (although other kinds of mistake are possible). All the agent knows is that the light was red at 11:01. However if the light does not change, the agent has to infer that the light does stay the same color for each time step. This can be done using a projection default like $LightColor(c, t) \hookrightarrow LightColor(c, t+1)$.

The problem with this approach is the amount of computation that is required to advance the facts in the KB to later times. There are likely to be many such facts in the KB and it becomes very expensive to have to infer that they do not change. All three approaches have the same problem.

Another problem with these approaches is that the KB can get very large with all the history of the agent explicitly represented at all times.

Indexical formulas

An alternative approach is to take all formulas in the KB to be indexical with respect to time. If a formula $LightColor(red)$ is in the KB at 10:00, then that means that the agent believes that $LightColor(red)$ is true at 10:00. As time moves to 10:01 and this formula is still in the database, the agent believes that the light is red at 10:01 too. The advantage of this approach is that we do not need to infer that things stay as they are. The assumption that things usually do not change is built into the system. The disadvantage is that since we do not explicitly project the beliefs, we may have false beliefs. For example, if the light changes at 10:01, the KB will still contain $LightColor(Red)$ which is a mistake. The mistaken beliefs can lead to mistaken actions or generate contradictions that need to be reasoned about. In this case, we also know that $LightColor(Red) \leftrightarrow \neg LightColor(Green)$ and observe $LightColor(Green)$, we would obtain a contradiction.

Another shortcoming of this approach is that it does not keep a history of the beliefs in the KB as the previous ones do. This is not a problem with Alma since it does maintain a history of the reasoning that is accessible from within Alma.

The gain in efficiency of the indexical approach depends on the extent to which formulas change truth value. If formulas change their truth value with great frequency, we might be better off with explicitly computing their truth value at every step. It seems however that most formulas will retain their truth value for reasonable lengths of time. This approach also simplifies the syntax of the logic and the inference rules required. This is the approach adopted here and is in keeping with the ideas behind active logic. Indexical representation are generally more efficient than explicit ones [164, 19].

10.2.2 Representing actions and observations

Since the agent is situated in time, it should be aware of events occurring in the world. The time at which these events occurs is important for determining the truth of the beliefs of the agent. If the agent does not know when it observed that the light is red and when it observed that it is green, it might have some trouble knowing what color the light is now. This makes mistake determination hard too. The representation for events does therefore need to include temporal information. We do this by adding a time argument to events.

Observations are represented as $Observe(\phi, t)$ where ϕ is the fact observed and t is the time at which it was observed. This is not necessarily the time at which the fact became true. For instance, the light color can change at 10:00 but the agent may only observe the new color at 10:01. Examples: $Observe(Light(Green), 10:00)$, $Observe(Start(race), 10:01)$.

Actions that the agent undertakes have to be expressed in the KB. We assume this is done in a similar way to the Carne predicates so that we have $Doing(A, T)$, $Done(A, T)$ and $Error(A, T)$ which represent that the action has been started, has completed or has failed, respectively.

10.2.3 Representing the relation between events and beliefs

These events, including observations, need to be converted into beliefs about the world for the agent to reason about them. From the above observation, we need to be able to obtain $LightColor(Green)$ in the KB. This could be done through a default like

$$Observe(\phi, t) \hookrightarrow \phi$$

We assume here that the observations of the agent are usually veridical. If the conditions are such that this is not the case, we need formulas expressing these conditions. An example of this is when the ambient color changes as earlier.

These are defaults because there are situations where what is apparent from our observations is not true. For instance if the agent has its red and green receptors switched, if it sees green, it is really red. We could express that as $Observe(LightColor(Green), t_i) \wedge RedGreenSwitch \hookrightarrow LightColor(Red)$. This default should be preferred to the previous one and that preference can be asserted in the KB, or it can be derived through specificity.

This is just like any other default and belief. Note that this representation implies that the fact that ϕ has been observed at t cannot be mistaken but the inference from the raw observation to facts about the world can be mistaken.

10.2.4 Preferences

Preferences have been defined previously as relations between defaults and are asserted in the KB as part of the description of the domain and cannot be changed or augmented later. In the current situation, we need to relax some of these constraints.

In the example about the traffic lights, the light changed from red to green and we would like the agent to prefer the later observation. This can be most easily described by a preference for the later observations than preferences for defaults. Note that we would like to state that if two observations conflict, the agent is to prefer the later one. If we were to assign preferences to defaults, we would have to do so using the observation–belief default in the previous section. The representation then gets more complex. The preference for the later observations is also a default since we do not always prefer the latest observation.

We therefore generalize the notion of preference to allow preference to be a relation between any two formulas, not just defaults. The preferences can be computed by the agent using default and non-defeasible formulas. The use of defaults means that the preferences can be themselves mistaken and therefore lead to a mistaken response. When this is detected, the preference has to be repaired and following that, the initial contradiction (or possible contradiction) has to be repaired too. This is greatly simplified by the explicit assertion of mistakes and the reasoning to the resolution of the mistakes. If these are not represented, it becomes harder to undo mistakes that may involve several previous undoes.

10.2.5 Properties of predicates

As seen above, we would like to express that for some observable facts, a later fact is preferred to an earlier one. For instance, if the traffic lights are observed at 10:00 to be red and at 10:02 to be green, then we want to prefer the latter to the former. This need not always be the case, however.

We could assert formulas to assert the preferences for each predicate of that sort, for example

$$Observe(LightColor(x), t_i) \wedge Observe(LightColor(y), t_j) \wedge t_i < t_j \wedge x \neq y \hookrightarrow Prefer(Observe(LightColor(y), t_j), Observe(Lig$$

This would be part of the domain description and would be part of the description of traffic lights. But doing things this way is not efficient since we are likely to have a large number of similar formulas for many of the things we can observe. Further, this seems to be a property of observations rather than of the particular fact observed.

Since this is likely to be a property of a large number of predicates, a more economical solution is to assert that as a property of the predicate and use that to derive the above formulas. This is not necessary, but makes representing the domain more convenient.

$$PreferLater(Observe(x, t))$$

$$PreferLater(\phi) \wedge \phi(\overline{x_i}, t_i) \wedge \phi(\overline{x_j}, t_j) \wedge t_i < t_j \hookrightarrow Prefer(\phi(\overline{x_j}, t_j), \phi(\overline{x_i}, t_i))$$

10.2.6 Plans, intentions, actions

Plans, intentions and actions have to be represented in the agent if it is to reason about these. We could use any representation of intentions, plans and actions that can be expressed within this framework for example that in [151]. Although we could reason about executing the plan and even about building the plan in the logic, this sort of computation may be better done in a specialized plan execution or planning system.

Assuming that there is an external plan execution system that does the planning and plan execution that the agent reasons about, the degree of detail of the logical representation must be determined. This relates to the choice of whether to represent all steps of a plan or whether some plans although they consist of many steps, can be seen as atomic from the point of view of the logic. The choice made will determine the range of mistakes the agent can handle, the degree of control of the logic over the agent's action and also the efficiency of the reasoning. This is another instance of the flexibility/efficiency tradeoff. Having a large amount of detail about the plan and its execution enables the logic to react to a potentially larger range of mistakes, and to repair mistakes more effectively however the amount of reasoning required will be greater too.

10.2.7 Truth and beliefs

To detect mistakes and to reason about them, the agent has to distinguish between its past beliefs, what it now takes to have been true in the past and its current beliefs. The agent might have thought, a year ago, that salon.com was a good investment and bought stock in it. With the company now delisted, it realizes that it had been wrong at that time. It needs to distinguish between its belief a year ago, and what it now believes was the case a year ago.

As mentioned above, we will represent a current belief that ϕ simply as ϕ. A belief that ϕ at time t_0 is represented as $Bel(\phi, t_0)$. A belief that $\neg\phi$ was in fact true at t_0 is represented as $Holds(\neg\phi, t_0)$. Note that this seems to revert to the explicit representation of time. This is the case but this representation is used for past facts, and mainly for those past facts that were mistaken.

Bel facts cannot be mistaken when they are about past beliefs–they only involve a lookup of the history to verify what the agent believed at that time. $Holds$ facts, however, are subject to change. At t_0 the agent might have believed ϕ but it can later change its mind and believe that at that time ϕ was not true. This allows the agent to assert $\neg Holds(\phi, t_0)$. This belief can itself be mistaken and the belief now of the truth at that time can change again so that we get at t_5, $Holds(\phi, t_0)$. This generates a contradiction that must be resolved, presumably by preferring the later information.

We now present some useful relations between these formulas.

First of all, usually anything that is in the database now is something that the agent believes is true now so:

$$\phi \wedge Now(t_i) \hookrightarrow Holds(\phi, t_i)$$

Recall that $Now(t_i)$ is available in the active logic.

Usually what is believed is taken to be true

$$Bel(\phi, t_i) \hookrightarrow Holds(\phi, t_i)$$

However these two are rather weak defaults and if there is any evidence that ϕ does not hold, these defaults are defeated.

An axiom that does hold though is the following:

$$Now(t_i) \wedge \phi \rightarrow Bel(\phi, t_i)$$

This represents that the agent is not mistaken about the history of its reasoning. If ϕ is in the KB at t_i, the agent believes ϕ at t_i.

As mentioned above in the discussion on representations of time, $Holds$ requires an explicit expression of inertia which is not necessary if we represent facts indexically:

$$\forall t \; Holds(\phi, t_0) \wedge t_0 < t \hookrightarrow Holds(\phi, t)$$

This too is a rather weak default and a change in the world will change what holds.

The agent can believe only one of ϕ or $\neg\phi$ holds at any time so we have:

$$Holds(\phi, t) \rightarrow \neg Holds(\neg\phi, t)$$

And also

$$Holds(\neg\phi, t) \rightarrow \neg Holds(\phi, t)$$

The converse of this does not hold though since the agent might not know anything about some formula. We could have $\neg Holds(\phi, t) \wedge \neg Holds(\neg\phi, t)$. This is not incoherent since $Holds$ represents the *belief* of the agent as to what was the case, not what was the case.

The time always holds at that time:

$$Holds(Now(t), t)$$

These relations allow for contradictions that involve $Holds$ formulas: that is contradictions about the agent's beliefs about what held in the past.

$Holds$ is useful not only to help identify mistakes but also to reason about facts (and not the agent's beliefs) in the past. For instance, the agent may have believed that the light was green at 10:00 but now realizes that it was red, so it can reason that it was illegal for it to have crossed the road at that time and that doing so was a mistake. This suggests that the formulas used for reasoning indexically should be usable to reason about the past through $Holds$. We can transform formulas about the present to formulas about the past, which we call the *Holds-analog* of the formulas using the following procedure.

Given a formula ϕ,

1. If ϕ contains $Now(t)$, if there is no $\forall t$ at the front of the formula, add it. For each literal L of the formula that is not a $Holds$ or $Observes$ or Bel or Now formula, convert that formula into $Holds(L, t)$.
2. If ϕ does not contain $Now(t)$, then for a new variable s, add $\forall s$ in the front of ϕ. For each literal L of the formula that is not a $Holds$ or $Observes$ or Bel or Now, convert that formula into $Holds(L, t)$.

If a formula is true, then its Holds-analog is true too. For instance, $Light(Red) \rightarrow IllegalToCross$ becomes $\forall t\ Holds(Light(Red), t) \rightarrow Holds(IllegalToCross, t)$. Given the Holds-analog, the agent can reason about the past in the same way as it reasons about the present. This enables the agent to correct its past views.

10.2.8 Expectations

Since the agent is supposed to reason about actions and execute them, it needs to assert and reason about expectations about the results of its actions. If the agent executes an action at 10:00 with the expectation that ϕ is true at 10:03 and it observes that $\neg\phi$ is the case at this time, the agent should realize that this means that the action failed. This should then trigger repair actions.

The representation for expectations must facilitate this. We represent expectations using the $Holds$ predicate so that if the agent asserts that $Holds(\phi, 10:03)$, and it is now 10:00, this is interpreted as an expectation by the agent that ϕ will be true at 10:03, or that come 10:03, ϕ will be true. This relation can be expressed as:

$$Holds(\phi, t_i) \wedge Now(t_i) \hookrightarrow \phi \qquad (10.1)$$

Using formula 10.1 will, by default, result in ϕ being asserted at 10:03. If the agent's prediction is correct, this assertion reflects the world. If this prediction is not correct, it will observe or infer from other observations that $\neg\phi$ and this can be recognized as a failed prediction.

10.2.9 Mistakes

Recall the definition of mistake: if $Bel(\phi, t) \wedge \neg Holds(\phi, t)$ then $Mistake(\phi, t)$.

Mistakes now need to be explicitly represented because unlike the case of mistaken beliefs, we do not have the regularity of the situations which enables mistake handling to be compiled in procedures. We need to represent the mistaken object–a belief, intention or action, and the time that this mistake was made.

The explicit representations also facilitates reasoning about multiple mistakes and mistakes of mistakes as described above.

10.3 Reasoning

An agent that forms beliefs and acts in an uncertain and dynamic world needs to have processes that reason about various aspects of the representation presented above. Some of the more prominent kinds of reasoning are

- **Detecting and identifying mistakes** This is the first step in handling mistakes. The agent must minimally detect that there has been a mistake. The range of mistakes it can detect and the accuracy with which it can identify the cause of the mistake will determine its performance and flexibility.
- **Propagating mistakes** Once a mistake has occurred, it is useful for the agent to determine what other objects might be mistaken. The degree to which that is done will influence the coherence of the behavior of the agent. An agent which believes that ϕ was mistaken, yet does not believe that doing action A is not mistaken A even though the only reason to do it was ϕ can seem to act incoherently.
- **Undoing the consequences of mistakes** Once a mistake is propagated, the agent should consider a repair of the consequences of the mistake.

Once again, agents can have these sorts of reasoning processes to various degrees. In particular situations, a very limited capability in one or more of the classes can be adequate and can indeed be the best solution considering the resources available and the time available for reaction. We discuss some of these factors in greater detail below in the context of our agent design.

10.3.1 Reasoning about intentions, plans and actions

Since the agent plans and acts, an issue is whether the behavior of the agent is to be generated from logical computations. Since the plans, intentions and actions are to be represented in the logic, this seems to be possible. However, deciding what actions to take based on logic is likely to be slow and inefficient. This is especially problematic in domains where the agent needs to respond to changes in the world with reasonable speed.

A solution to this problem may be to execute the plans that the agent has in a plan execution system but have the actions and decisions taken by the system reflected in the logic and giving the logic the possibility to intervene in the operation of the plan execution for instance by adding or deleting intentions in the plan execution system. The logic should be kept informed, for instance, of the adoption of plans, the start of execution of actions, whether these actions succeed or fail and so on. Some details of that can be found in [151] and can make use of abilities as that of Carne discussed above. This approach gives the speed of the plan execution system with the flexibility of the logic. As long as there are no mistakes, the plan execution system is in control and the computations are efficient. Once there is a mistake, the logic can come in which enhances the flexibility of the system to deal with unexpected events. If the plans are designed to deal with a set of errors, the logic can then only come in in cases of unanticipated errors or when it can infer that an error has occurred before its consequences are manifested in the world.

10.3.2 Reasoning about what holds

We have seen that for each formula describing the world, we will also have the Holds-analog of that formula. The reasoning with the Holds should be identical to the reasoning without it. To do that we have a set of inference rules similar to those for the first order logic that instead of being about formulas ϕ are about formulas $Holds(\phi, t)$ and apply only when the times are identical. The modus-ponens rule would look like:

$$\frac{Holds(\phi, t), Holds(\phi \rightarrow \psi, t)}{Holds(\psi, t)}$$

We will also need to detect contradictions in the Holds formulas. However, no special rules are needed since from $Holds(\neg phi, t)$ we get $\neg Holds(\phi, t)$ from above.

10.3.3 Overview of reasoning about mistakes

Just as the representations for dealing with mistakes in this case have to be enhanced from those in the previous case, so do the reasoning procedures. The overall view is similar to the one before except that now, we make more of the actions of the procedures explicit and record them in the KB so that they can be reasoned about later.

The high-level view of handling a mistake is as follows:

1. The fact that there is a mistake is signaled by a contradiction.
2. The contradiction is resolved if possible by preferences.
3. The cause of the mistake is determined and is asserted.

Once these mistakes are asserted, the response to each mistake comes into play. This involves the following:

1. Undo the mistake.
2. Verify whether the consequences of the mistake are mistaken and assert so if that is the case.
3. Verify whether the mistaken formula was also mistaken at a previous step and assert so if true.

Undoing a mistake depends on the kind of mistake we are considering. Beliefs, intentions and actions have different ways of being undone. The information about how to undo intentions and actions has to be asserted as part of the domain description. Some details of the process follow.

10.3.4 Detecting and identifying mistakes

The logic depends on direct contradictions to detect that there has been a mistake. We need to ensure however that contradictory information does get represented as direct contradictions in the KB. The sources of mistakes can be beliefs and observations that are not consistent, or actions or expectations that fail. Another source of information about mistakes is the failure of actions. This can be noticed either by a failure notification by the action execution mechanism, or by an observation that the predicted consequences of the action do not hold in the world. Each of these combinations should be detectable as a direct contradiction.

Mistakes among the current beliefs are easily detected with the contradictions among indexical representations. Mistakes in the past are detectable through contradictions among the $Holds$ assertions and failed actions can be determined through contradictions between the expectations of executing the action and observations when these are connected to facts about the present or the past. Mistaken actions can also be detected through the assertion of $Failed$ predicates as a result of doing the action.

Distrusting contradictands

As soon as a contradiction is detected, the contradictands and their consequences are distrusted by Alma so that they do not derive new contradictands. This is done by asserting $distrust(\phi)$ for the formula. The contradiction detection rule makes the formulas distrusted and propagates the distrust to its consequences. This prevents the bad effects of this contradiction from spreading into the KB and the logic can then have some time to reason about which contradictand is true without more mistakes being made. This is similar to the reaction in the mistaken beliefs case.

Preferences

Identifying the mistake in this case is harder because the preferences are no longer fixed and the logic cannot prevent contradictions involving formulas with preferences between them. The preferences may have to be computed *after* a contradiction is detected. This makes the contradiction resolution rule differ from that in the case of mistaken beliefs. While it is in principle possible to compile the various inferences needed to identify the contradiction into complicated preference formulas for the observations, this is likely to result in a large number of complex formulas.

In the mistaken belief case, we limited the kinds of contradictions that we would consider by assuming that all preferences are known beforehand. This results in the agent not applying a less preferred default if a more preferred one has been applied, and also removing a less preferred default application if a more preferred default is being applied. The result is that none of the contradictions obtained involved any pair of defaults with preferences between them. This is not necessarily the case anymore. A contradiction can be derived even though there is a preference relation between the defaults involved. If that was not known at the time that the defaults in question was being applied.

The logic needs to be able to compute some preferences at the time we get the contradiction, especially for preferences between observations that occur at different times. The preferences can be inferred from axioms and defaults that describe the domain so that this preference is flexibly specified. This is more conveniently computed using the backward search facility of Alma. Once there is a contradiction that involves a group of defaults, we can try to prove a preference relation between every pair of defaults.

Identifying the mistake

Once the contradiction is asserted in the KB, the logic tries to find preferences that will solve the problem. This is done by looking for preferences between pairs of defaults, each of which supports one of the contradictands. Since we have a record of the reasoning and the derivations are available, this is possible although possibly complex. Once this is found, the identity of the preferred contradictand is asserted in the KB. In the mean time neither both contradictands are distrusted and can draw no new conclusions. The KB also contains the fact that both contradictands cannot be preferred. This causes a contradiction if there are conflicting preferences and the presence of that contradiction triggers further action–this is then meta-meta reasoning. Active logics are designed to represent and reason with multiple meta-levels so this is not a problem. The less preferred formula is taken to be the cause of the contradiction and this too is asserted in the KB. The cause is asserted to be mistaken and inference rules to handle the mistake come into play.

However, we need not always find one preference in the defaults. There maybe none, or there maybe conflicting ones.

No preferences If there are no preferences to resolve the contradiction, then there is no resolution and the consequences of the contradictions remain distrusted.

Conflicting preferences If there are conflicting preferences, a contradiction will be generated between the preferred contradictand formulas and these preferences will have no effect on the original formulas until that contradiction is resolved.

10.3.5 Undoing mistaken actions

Actions can be divided into actions taken in the outside world and actions on the KB. These actions may need to be undone in the event of a mistake and the undoing may also need to be undone. The approach taken here is to assert in the KB all the actions taken and to use the record of the derivations and of the consequences of formulas to undo these actions.

Beliefs and intentions are represented as formulas in the KB and so undoing an addition of a belief consists in deleting the belief. Intentions may additionally have consequences outside the logic and these are explicitly undone in a similar way to undoing actions. The undoing of external actions is specified in the domain description and undoing them consists in doing the repair actions specified.

Undoing KB actions

We now discuss the actions the logic does in the KB and the way these are undone. The actions of interest are additions, deletions and distrusts.

Addition Adding a formula to the KB is done automatically by the inference rules and is not explicitly represented in the KB. When an inference rule application needs to be undone, we simply delete the conclusion of the rule from the KB and assert that that formula has been deleted. The application of the rule and its results is recorded in the KB.

Deletion When a formula is added to the KB of the form $Delete(\phi)$, ϕ is removed from the KB, formulas of the form $Delete(\psi)$ for each ψ that derives from ϕ are added to the KB. If there are multiple derivations for a consequence of the mistaken formula, the derivation from the mistaken formula is removed. If there are no more derivations left, the consequence itself is removed. This approach is similar to that used in TMS [43].

Deletions are undone when $Undo(Delete(\phi))$ is added to the KB. This results in ϕ being added and $Undo(\theta)$ being added to the KB where θ are formulas of the form $Delete(\alpha)$ that derive from $Delete(\phi)$.

Distrust When a formula of the form $Distrust(\phi)$ is added to the KB, ϕ is removed from the set of formulas that can be used for inferences and formulas of the form $Distrust(\psi)$ are added to the KB for each ψ that derives from ϕ through one inference rule application.

If $Undo(Distrust(\phi))$ is added to the KB, the reverse process occurs: ϕ is made available for inference and for each formula of the form $Distrust(\theta)$ that derived (in one step) from $Distrust(\phi)$, $Undo(Distrust(\phi))$ is added to the KB. In case we distrust a deletion or a distrust, we undo these actions.

Undoing external actions

Since the actions done by the logic are few and predefined, we can have inference rules to undo them. For actions taken in the outside world however, the undoing has to be specified in the domain specification.

Recall that the formula that triggers an action A at time t is represented as $Do(A, t)$. An inference rule will cause A to be executed at t and asserts the status of the action in the KB (Doing, Done, Failed). A can be a primitive action or a plan. The expected outcome of executing the action is computed by the logic and is also asserted in the KB.

Undoing an action can involve the doing of some other action. For instance:

$$Undo(Do(DeleteFile(one.tex), 10:00)) \rightarrow Do(Recover(one.tex, 10:00))$$

To undo the deletion of a file at some time, we attempt to recover a copy of the same file at that time. Formulas like that have to be asserted in the KB as part of the domain axiomatization.

If that recovery is later found to be mistaken, we end up with $Undo(Undo(Do(DeleteFile(one.tex, 10:00))))$. This results in that file being deleted once again.

10.3.6 Propagating mistakes

In the above, we have considered how to detect and locate the cause of the mistake. The cause is labeled as being a mistake. In this section we discuss propagating the “mistake” label to other formulas and to other times. We will express this as an inference rule to propagate mistakes. Further, the determination that there has been a mistake can itself be mistaken so that the logic should be able to undo the effects of that mistake too.

Propagating to consequences

If ϕ is mistaken and we know that ψ was derived from ϕ, if there are no other reasons for believing ψ, that too should be mistaken because the logic believes that ψ holds but it does not. This too is similar to the TMS approach. In this way, we can propagate mistakes to consequences so these consequences too can be undone. Note the difference with the NMR case where we deleted all the consequences immediately. In this case, the deletion is done more deliberately with each step of the process recorded in the KB because it might involve undoing old mistakes or external actions.

Propagating to the past

Once we know that there is a mistake at some time, in addition to propagating the mistake to its consequences, we are interested in knowing whether there was the same belief at an earlier time and whether this was then mistaken.

Computing $Mistaken(\phi, t_i)$ requires the logic to find $Bel(\phi, t_i)$ and $\neg Holds(\phi, t_i)$. While it is straightforward to compute Bel, computing $Holds$ requires the logic to reason about what it now believes to have been true at t_i, even if it did not believe it at that time.

The approach is to use the $Holds$-analog axioms and rules to infer whether $\neg Holds(\phi, t_i)$. This will be the case if $Holds(\neg\phi, t_i)$ and ϕ is not preferred to $\neg\phi$. Since $Holds(\neg\phi, t) \hookrightarrow \neg Holds(\phi, t)$, deriving $Holds(\neg\phi, t)$ is one way of proving $\neg Holds(\phi, t)$.

While the $Holds$ axioms and rules allow us to infer whether the consequences of an axiom or default hold if the premises do, it does not help for those formulas that are not derived from others: those that are axioms and are in the KB at the start of the computation and those that are observed in the world.

Some of the difficulties associated with this is that some of these formulas that are true at some point may become false later and conversely, formulas that are true now, may have been false earlier. This makes the computation of what holds dependent on knowing what changed in the world.

If, for instance, we know at 10:00 that Tweety is a bird and at 10:10 come to know it is a penguin, we will initially think that Tweety flies and later think that Tweety does not fly. We were mistaken about Twety flying at 10:00 before we learned that Tweety is a penguin because we believed a fact that did not hold. The fact that we learned that Tweety is a penguin later and that we could not have possibly derived that Tweety is a penguin initially does not change anything to the fact that we had made a mistake. Any actions we might have taken based on the mistaken belief, for example dropping Tweety off a cliff, is likely to have bad consequences. Therefore it is not sufficient to know when we knew the relevant facts but also when these facts were true in the world. The logic needs to know that the fact that Tweety is a penguin is not one that suddenly becomes true after the fact that Tweety is a bird.

On the other hand, there are situations where a fact does change in the world and where we can assume that the logic is aware of the change as soon as it happens. For instance, the logic might see that the light is red and reasons that it should not cross the street. Later it notices that the light is green and so it can cross. This does not mean that it had made a mistake in not crossing earlier because the light did change from red to green because at that time it thought it was red and it was indeed red.

For the agent to be able to reason in this way about what was true in the world at what time, it needs to know how facts change in the world. For instance, that penguinness is not something that typically changes; or that traffic lights typically change and that we typically notice it the moment they change. This sort of information needs to be specified when describing the domain.

The fact that penguinness tends not to change can be expressed as

$$\forall x\, t\ Penguin(x) \rightarrow Holds(Penguin(x), t)$$

In the case of observations, we want to express the fact that if the light has been observed to change from red to green, then before the change it was not green.

$$\forall t_i\ Observe(Light(Red), t_i) \wedge Observe(Light(Green), t_j) \wedge t_i < t_j \rightarrow \forall t_k\ t_i < t_k < t_j \rightarrow \neg Holds(Light(Green), t_k)$$

Before we observed the light to be red, it could very well have been green, so we don't want to express that the light cannot be green before we observe it.

There are other similar properties that we may want to express of various classes of predicates. This can be unwieldy if expressed separately for all the predicates. If the logic allows quantification over predicate names as mentioned above, then such sentences can be expressed more compactly.

These sorts of axioms enable us to determine whether a formula holds at some time in the past and therefore whether there was a mistake.

10.3.7 An example

The following examples can clarify this approach. Assume that at 10:00 the agent observes that Tweety is a bird and therefore concludes that it flies. It later observes that Tweety is a penguin and decides that Tweety does not fly and that it was a mistake for it to believe that Tweety flies. The determination of the mistake is done using the contradiction and a preference for penguins not flying. We can represent this as follows:

Time	Event	Belief
10:00	$Observe(Bird(Tweety), 10:00)$	$Bird(Tweety)$
10:05		$Flies(Tweety)$
10:10	$Observe(Penguin(Tweety), 10:10)$	$Penguin(Tweety)$
10:15		$\neg Flies(Tweety)$
10:20		$Mistake(Flies(Tweety))$

The agent needs to determine whether the inference that Tweety flies at 10:05 was mistaken. It believed $Flies(Tweety)$ at that time. To determine whether that held, it verifies whether, given what it knows now, Tweety did fly at 10:05. A possible derivation is as follows:

1.	$Observe(Penguin(Tweety), 10:10)$	From observations
2.	$\forall t, \phi\ Observe(\phi, t) \hookrightarrow \phi$	
3.	$Penguin(Tweety)$	1, 2
4.	$\forall xt\ Penguin(x) \rightarrow Holds(Penguin(x), t)$	Axiom
5.	$Holds(Penguin(Tweety), 10:05)$	3, 4
6.	$\forall x\ Penguin(x) \hookrightarrow \neg Flies(x)$	Default
7.	$\forall x\ Holds(Penguin(x), t) \hookrightarrow Holds(\neg Flies(x), t)$	6
8.	$Holds(Flies(Tweety), 10:05)$	5, 7

The logic can therefore conclude that it was a mistake to believe that Tweety flies at 10:05. Note that $\forall xt\ Penguin(x) \rightarrow Holds(Penguin(x), t)$ has to be added in the domain description to assert that if anything is a penguin, it is always one.

10.3.8 Repair

Repairs of mistakes involves undoing the effects of the mistakes. Information for undoing actions in the world are specified as part of the domain description. Logical actions on the KB are undone using rules as mentioned above.

Once the logic has determined that a belief was mistaken, it needs to repair it. The action can be undone by asserting $Undo(A, t)$ and letting the appropriate domain-specific inference rules and axioms come into play. $Undo(A, t)$ can be asserted in the KB simultaneously with the propagation of the mistake. It can be added to the KB once a formula is found to be mistaken.

In the case of mistaken actions, the undo predicate applied to the action call will cause the appropriate inference rule to undo the results of the action if it has already been executed according to the undo information that is part of the domain description.

10.4 Related work

The problem of agents that act in uncertain dynamic worlds has been addressed in work in planning and plan execution. We review some of these systems and discuss their approach to the factors that we found are useful in handling with mistakes.

10.4.1 CNLP–Conditional planning

Conditional planners, as opposed to classical planners [56], recognize that the world is uncertain and that the planner has limited information about the world. The uncertainty is present not only in the world, but also in the results of actions. These factors make failures in plans inevitable. The solution that conditional planners implement is to build plans that have actions that can observe the world. Based on these observations, different paths can be taken. The observations could, for instance, verify that the conditions that a prior step of the plan was supposed to have set up hold. If they don't, that implies a mistake in the plan and the plan can be designed to produce sequences of actions to deal with that mistake.

CNLP [139] is such a conditional planner that develops plans that account for foreseen uncertainties. It was seen as the first step towards the development of a decision theoretic non-linear planner. The observations made during plan execution are to be used to select which actions are executed. The active detection of failures means that replanning at runtime for those errors is not necessary.

The plans that CNLP produces are trees with the outcome of each observation generating a subtree that gets to the goal given that observation. The result of this is that the size and complexity of the plans generated increase exponentially with the number of observations. This problem can be mitigated by skipping unlikely combinations of observations.

Since CNLP is a planner, we will discuss the behavior of an agent executing a CNLP plan.

- **Beliefs** Beliefs are not explicitly represented. They are implicit in the state of the plan. As new information is observed and new actions taken, the state changes. So we can see the beliefs as being in time. However there is no history.
- **Actions** Actions and plans are represented in the structures of the plan. They cannot be reasoned about by the executing agent. Intentions are not explicitly represented either.
- **Events to beliefs** The observations cause an immediate change in the state of the system. This can be seen as the system believing the observations.
- **Beliefs to truth** There is no notion of truth separate from that of beliefs.
- **Mistakes** From the above, mistakes are not explicitly represented. They are implicitly detected and acted upon.
- **Expectations** No expectations are explicitly represented in the system. However, the observations done and the branch in the plan reflects expectations in the planner as to the state of the world.
- **Detection and identification** Mistakes can be detected by observations. Not all observations imply a mistake has occurred and the mistakes are not identified any further.
- **Propagation** Mistakes are not explicitly represented and are not propagated either.
- **Undoing** The results of a mistake can be undone through the branch taken. This can be done if needed by the planner but is not necessary.

All possible states of the world and outcomes of actions have to be handled in the plan. Repairs to mistakes are represented in a branch of the plan. Since all the reasoning and explicit representation is done in the planner, this leaves the executor with little flexibility. If the world and its contingencies are just as predicted by the planner, the system will work efficiently and solve all problems. However, if there is an unexpected event, the agent executing the plan is likely to fail if it has no further resources. Note that this can be a first step in a layered approach that has means of dealing with unexpected failures.

10.4.2 C-BURIDAN–probabilistic contingent planner

C-BURIDAN [45] is a probabilistic planner based on BURIDAN [96] which senses the world and takes actions depending on the outcome of the sensing action, in a way similar to CNLP. The major difference with CNLP is that the world is described with a probability distribution over the states and the actions have probabilistic outcomes. The goals are similarly achieved at some degree of probability. Sensing actions also produce observations with some probability attached which allows the modeling of noisy sensors. The plans produced by C-Buridan also allow paths to split at conditions but contrary to CNLP, these can merge later which reduces the size of the plan produced.

C-Buridan also makes a distinction between the changes an action makes in the world and the changes it makes to the knowledge of the agent. Actions made in the world change the probability that the consequences of the action are true. Observation actions however, do not change the world but provide evidence for the state of the world. This changes the probability of variables of interest through Bayes's rule [136].

C-BURIDAN has very similar characteristics as CNLP as it is also a conditional planner. The main point of interest here is that the probabilistic representation makes explicit the uncertainty in the world. In our approach, uncertainty is represented as defaults. In this case, the degree of uncertainty is taken into account while building plans. The availability of probabilistic information can allow for finer-grained judgments of the source of mistakes. It also facilitates diagnosis of problems though the use of Bayesian nets. However C-BURIDAN does not use these facilities to reason about mistakes, it is still a conditional planner which produces a plan that is meant to be executed at run time with no further reasoning.

10.4.3 MRG–planning with failure

MRG [186] is a system that allows one to represent and reason about plans and to to build a wide range of kinds of planners. The language of MRG allows failure and success of plans to be expressed and reasoned about which allows control structures that react to failures. The plans in this case can contain the basic operations that deal with failure. MRG can produce planners that are reactive–that react immediately to an event such as a failure, or conditional planners, or deferred planners that interleave plan formation and execution.

The main construct for dealing with failure is the expression $iffail\ \alpha\ then\ \beta\ else\ \gamma$. This says that if α fails, the plan is to do β else γ. This is used to define other constructs that express for example, that the planner should try an action, and if that fails, to fail the plan or to execute an action only on failure of a previous action.

Giunchiglia et al [68] aim to provide a theory to represent and reason about actions and plans that are not guaranteed to succeed. The plans and failures that are expressed in the MRG language are reasoned about in the MT* language that is a meta-theory of MRG. Plans in MRG become terms in MT*. The language of MT* relates plans to the conditions that hold after the plan is executed. MT* allows one to assume that an action succeeds or fails and compute the consequences of the success or failure of the plan.

MRG allows the development of planners that handle failures in a variety of ways. Failures are explicitly represented and that allows them to be handled more flexibly than purely reactive systems. Meta-reasoning is also possible in MT* which should make the system more flexible. However, there does not seem to be much reasoning about failures at run-time. This makes the agents that execute plans produced by MRG similar to the above cases. Reasoning about the failure to find its source, for instance, does not seem possible. MT* does reason about failures and plans but is a separate meta-language and is not available to the agent when the plan is executing. The focus here is more on planning and reasoning about plans produced rather than executing plans and reasoning about the execution as we do.

10.4.4 ESL–cognizant failures

ESL [62] is a language for encoding execution knowledge in agents. It coordinates the actions of a reactive component and a deliberative component that generates plans. ESL handles contingencies based on the concept of *cognizant failures*. This states that systems should be designed to detect failures when they occur so that it can respond appropriately.

There are two main constructs for handling contingencies in ESL: a way to specify that there has been a failure (FAIL) and a way to associate a kind of failure to a recovery procedure. *Guardian* tasks monitor constraints and generate a failure if the constraints are violated. Calling FAIL starts the appropriate recovery procedure. Some possibilities for recovery are retries and aborts. Failures can be propagated to more general recovery procedures if they cannot be handled at the level they are called.

ESL is similar to the conditional planners in that it handles errors when they occur with precomputed responses and just like the previous systems, there are specific conditions that the agent monitors to detect failure. The knowledge of what can be a failure is compiled into the choice of observations to be made.

However ESL has interesting features that seem to make it more general. Failures are explicitly represented by the FAIL form, although the failures possible are preset by the planner. The response to the failure can vary according to the situation since recovery procedures that fail propagate the failure to more general handling procedures. There is therefore some sort of propagation of errors, although not in the same sense as we discussed. This can make the range of errors handled larger than if the system had just one response to errors and adds some flexibility to the response.

10.4.5 3APL–Failures and recovery

A general analysis of failures and some abilities needed to deal with failures are presented in [81]. The 3APL language facilitates building agents that handle failures.

The capabilities required for agents that operate in changing environments are seen to be: to monitor the environment, to deliberate, to accept goals and plan for them, to recover from failures and to modify plans in response to changes in the environment.

The reasons for failure are seen to be limited knowledge and limited control of the environment. Three kinds of failures are listed: opportunity failures, ability failures and plan failures. Opportunity failures occur when the current situation prevents the plan from succeeding. Ability failures result from the agent not having an ability that is assumed to be available. Plan failures occur when no plans can be produced or inadequate plans are produced to achieve the goals. The responses to failures are to replan, to repair the plan or to drop some of the goals.

The 3APL [80] language is an agent programming language meant for BDI [155] agents and represents goals, rules and beliefs. 3APL is meant to provide the basic architecture for an agent, with the possibility of different capabilities being built into the agent. It specifies the control, application of rules to goals and execution of actions. The rules are used to find ways to achieve goals and can modify the goals themselves. The control operates in a sense-plan-act cycle.

Failures or success of actions is assumed to be entered to the system through robot sensors. There are two mechanisms to deal with failures: the disruption mechanism and the interrupt mechanism. The disruption mechanism monitors a goal π_m. If a condition ϕ holds, the monitored goal is replaced by a recovery goal ϕ_r. In the interrupt mechanism, the monitored goal is not removed but is interrupted by a goal π_i.

This paper has an interesting discussion of the capabilities needed for an agent which takes a different approach to the one we take here which is more related to the representation and reasoning abilities of the agent. The analysis of the kinds of failures possible are also interesting and could be used to engage different forms of response, although that does not seem to be done.

There are similarities between this and the previous systems in that the mechanisms to deal with failures are limited. The conditions for failure and the goal that they relate to have to be explicitly specified initially. It does not seem possible for the system to make general observations about the world and from these reason that there may be a failure without these linkages having been precomputed by the designer of the plan. The system is a general framework in which various modules can be added so that there can be a module that reasons about failures and modifies the plans or goals as a response to that. It is not clear however, how well these can be integrated with the system.

10.4.6 Phoenix–Analyzing failure recovery

In [85], failure recovery components for the planner for the Phoenix system are described, together with procedures used to debug the planner. We will focus on the first aspect.

Failures are seen to be caused by actions not having their intended effects, changes in the world and inadequacies of the planner. Failures are detected as events that preclude the successful completion of a plan.

The purpose of failure recovery is to repair plans as efficiently as possible. This transforms the bad state the system is in to a good state. This requires the planner to have a model of what to do in every situation. Automated recovery from failures involves two layers of the planner: a reflex layer that uses pre-programmed actions and a deliberation layer that coordinates actions and replans the goals.

Failures are classified by what is known to have blocked the execution. There is no effort made to find the cause of the failure. The recovery effort tries various recovery methods until one works. The applicable recovery methods are selected from a library of methods based on the cause of the error as the system can determine it. The repair methods make simple changes to the plan, for instance retrying the action or trying a different action.

Similarly to ESL, this approach allows for the fact that the repair for the error may be mistaken by trying all the possible repairs until one succeeds. Although inadequacies in the planner can be seen as mistaken beliefs, the system does not explicitly reason with beliefs or detects mistakes from them or propagates them to actions. Here too, like in the previous approaches, the plan is supposed to contain all the information that is going to be relevant to the agent. There is no flexibility for the agent to do reasoning to preempt errors or discover new ones if these have not been precompiled into the plan by the agent.

10.4.7 CIRCA–Detecting and Reacting to failures

Different classes of "unhandled" states are classified in [7] which also describes a planner that tests for and responds to these states. CIRCA combines a planner with a plan execution system with a reactive mechanisms. The question addressed is to detect when the agent is in an unexpected state and how to react to that.

There is a variety of states that the planner can find itself in: handled-states from which the planner can reach the goal, dead-end states from which the goal is not reachable, removed states which are those states that the planner did not consider for lack of resources, and imminent failure states which are states that are unreachable given the plan but that would lead to failure if they were reached. The handled-states and the dead-end states are in the planned-for class of states. The agent needs to recognize when it reaches any of the other states that it did not plan for and respond to that.

CIRCA control plans are represented as cycles of test-action pairs (TAPs). So TAPs have to be built to recognize that the agent is in one of the unexpected states and to act to try to get to a planned-for state. Recognizing that it is in an unplanned state is done by a set of tests that is derived from the ID3 algorithm [154]. The various unexpected states are found during the planning process and their class determined. These are input to the ID3 algorithm which then gives the minimal tests for those classes of states. The response to unhandled states is to replan. When that is not possible, the agent fails.

This presents an interesting approach to detecting the class of failures and an interesting classification of failures. However, just as the other approaches, it assumes that the planner does know all the possible failures that can occur. The replanning on failure can be seen as reasoning when the failure occurs. The reasoning does not however find the cause of the mistake or propagate mistakes or tries to undo possible bad consequences of the mistakes. But that may not always be necessary.

10.4.8 Cypress–planning, reasoning and acting

The Cypress project [191] develops agents that operate in dynamic and uncertain environments. This requires the agent to deal with unexpected changes in the world. They should be able to adapt their activities to the new situations. Part of this is the ability to modify plans when they are running. The Cypress system provides a framework for building this kind of agent. Cypress is built from the SIPE-2 planner [192], the PRS-CL execution system [64] and the GISTER-CL system for reasoning about uncertainty [179].

The SIPE-2 planner can plan for conditional actions, resource assignments and can modify plans during execution. The GISTER-CL system does evidential reasoning that is used during generation.

The PRS-CL system represents the current beliefs about the world (in a DB), a set of goals and the intentions of the agent. It also has a set of predefined procedures that specify what to do in response to events in the world or when a new goal is added. The environment is observed through the DB. The procedures can test for conditions, wait for events and can modify the beliefs, intentions and goals of the agent. PRC-CL can also request that the planner modify plans.

A test goal fails if the condition tested does not hold and the executor does not have procedures to make it true. Similarly, an achievement goal fails if the condition that is to be made true does not hold and no action applies to do so. In cases of failure, the plan is to be aborted and a replan request is issued. In the mean time, those plans and intentions not related to the failure proceed as usual.

Note that the failure of a goal occurs after there are no procedures in the executor that will react to the failure. Beliefs are not seen to fail

This system seems the closest to an agent that can reason, act and repair its failures flexibly in the world. The most interesting features are the explicit representation of beliefs, the explicit representation of intentions and goals that can be manipulated by the plans themselves, and the possibility to apply reactive responses to failures as well as replanning.

As the systems seen earlier, the failures CYPRESS detects are through conditions in the world that are predetermined by the planner. It does not seem that it can infer mistakes from general observations. The system does not explicitly represent beliefs as mistaken either, although it could modify its DB in response to mistakes. The reasoning module is only used to build plans and does not directly affect the state of the executor. However it seems that an appropriate reasoning module attached to the DB could do the sorts of reasoning we are interested in and have direct access to the plan execution by modifying the intentions and goals of the agent.

10.4.9 Exceptions in agent systems

We now consider work about exceptions in agent systems rather than in single agents. [91] proposes that instead of individual agents handling errors in multi-agent systems, there should be specialized agents for that purpose. This is based on a claim that such a division of labor allows for the separation of generic and reusable exception-handling behaviors from the domain-dependent problem solving behaviors. The problem solving agents then do not need to handle failures but there needs to be a standard interface language and the agents need to be self-aware and self-modifiable.

For errors to be detected, there is a need for a model of the normal behavior of the agent. The exception-handling agents are informed of this and of the failure modes that are known to occur with that normal behavior. They then set up sentinels to monitor for these conditions. The failure modes are derived from a taxonomy of problem solving behaviors with associated failure modes. The exception-handling agent maps the problem solving agents to this taxonomy to derive their failure modes. These failure modes can have associated with them scripts that generate the sentinels that look out for those failures.

The failure is diagnosed by matching the symptoms that occurred to a taxonomy of possible diagnoses based on symptoms. These are heuristic hypotheses of the diagnosis and need not be correct. Once one or more possible diagnoses are found, plans are generated to deal with the error. These come from a database of known generic error resolution strategies. There too, many plans are possible. The system tries these strategies and backtracks if they fail.

This case is also one where the detection, diagnosis and reaction to failures is precompiled into the agent. The difference is that we now have a specialized and presumably general failure detecting and repairing agent that does not depend on the domain. An interesting aspect of this solution is that the failures are diagnosed before being repaired instead of immediately applying a repair based on some observed conditions. Another interesting aspect is that the problem-solving agents are required to be self-aware and self-modifying. It is not clear what these entail, but these are powerful abilities for the problem solving agents to have and may be used for these agents to repair their own failures too. The purpose of these abilities seems to be so that the failure agents can change the behavior of the problem-solving agents to correct failures that are detected. It is also unclear what happens if the failure agents themselves fail. This approach depends mainly on a well-defined taxonomy of agent architectures, probable failures, symptoms and diagnoses. So it can only be done with well known agents in well known environments. New failures don't seem to be handled by the system.

10.4.10 Broker teams for fault tolerance

[95] considers brokers in a multi-agent system that can handle failures by working as a team. Restorative maintenance goals are introduced that allow new brokers to be started when any one of them fails.

Brokers [38] are used to accept requests from agents, locate agents and route requests to the appropriate agents and so a failure of a broker has to be repaired quickly. The failures considered here could be any of the sorts of failures in distributed systems, including failures of the agent, of the network and so on.

The solution proposed is to use a team of brokers that allow recovery from multi-agent failures and maintain a specific number of brokers running even though some might fail. This is done by adding an appropriate plan to the library of plans available to the brokers.

The approach here is similar to several of the other approaches seen above. On detection of an error, a repair is done, seemingly without trying to diagnose the source of the error. In this case as in the previous one, failure solving behaviors are seen as just another plan that the agents apply. This is suitable as the errors do not apply to the agents that detect the errors themselves but to other agents. The fact that these agents are now in a team seems to make the ability of the agent to repair itself less important as long as not all of the agents fail at the same time because of an error in their specifications.

10.4.11 Summary

The above are an interesting range of systems that deal with failures with varying degrees of generality. The main difference between these systems and our proposal is that the systems do deal with incompleteness and uncertainty, but they seem to assume that there is no incompleteness in the knowledge of the planner. That is that the planner knows all that can be incomplete or uncertain in the world during execution and can take care of these conditions explicitly. If it does not, the plan is likely to fail. This problem seems to be noticed though in the conditional planners where there are no branches in the plan for conditions that are judged to be very unlikely. In these cases, the designers seem to accept that the agent will fail.

There are great advantages to be gained by the planner precomputing possible mistakes and responses to them and compiling these into a plan. The alternative for the agent to try to resolve these problems at runtime or by replanning at runtime is very costly for mistakes that can occur frequently. However, the cases that slip though should not be ignored.

The CYPRESS system does catch those cases by replanning when there are no possible responses to a failure. The replanning takes into account the current state of the world and therefore may come up with a plan if more likely to succeed. However this system, just like the others (except for the Klein system) does not attempt to find the cause of the problem and try to repair all the mistaken consequences. Once again, if the planner knows all the ways the world could be, this could be encoded into the response to the failure.

A problem is that the systems do not have an ability to reason about themselves. The MT* system does do meta-reasoning, but this is about the object system and does not occur during execution. CYPRESS can modify its own intentions and goals and could be seen as doing meta-reasoning, but that seems to be limited in the scope of what is possible because there is typically no reasoning but just the application of procedures which have limited flexibility.

The meta-reasoning ability of Alma on which our system is based allows meta-reasoning and a greater ability for the system to modify its beliefs, intentions and goals while it is executing actions.

Another issue is that of time-situatedness. All the systems we have seen are time-situated. However, they do not have a history. This limits the reasoning about mistakes that they can do. They cannot, for instance reason back in time to the cause of a current mistake and find the possible mistaken consequences of that. This ability can be important if the agent takes actions in the world as these are meant to do.

More basic than meta-reasoning or time-situatedness is the problem of reasoning. Most of these systems do no logical reasoning during execution. They simply follow their plans and do replanning in some cases. As mentioned before, we can see planning as a form of reasoning but it lacks the generality of logical reasoning. This can result in a lack of flexibility in the agent. The 3APL system seems to allow for reasoning while the system is executing, however it is not clear how that is to be done and what the results could be. We do not believe that logical reasoning should be the only form of reasoning that should be done. However, not doing any of it restricts the flexibility and the adaptability of the agents

These systems all have very attractive properties and are capable in more or less strictly defined conditions. However they seem to lack the flexibility needed to act in the world in a human-like way. One of the problems, we suggest is the lack of reasoning, and meta-reasoning in these systems. The solution is not to move to a 100% logical reasoning scheme without any of the procedural techniques discussed of theses systems. As mentioned earlier, he solution may be using logic as a safety net to catch those errors that precomputed procedures cannot handle, for example by adding our type of logical reasoning to the DB of the Cypress system.

10.5 Future work

The above was a rather detailed sketch of the design of an agent that detects and handles its own mistakes while reasoning and acting in a dynamic and uncertain world which the agent only knows incompletely. This requires more work to get to the point where one could implement such an agent and use this to validate the ideas presented here. In addition to this, more work is needed on the more general aspects of mistakes.

We have earlier provided our definition of mistakes and a few properties that were useful for developing agents that react appropriately to mistakes. However, it is not likely that these are all the properties of interest. We need to do more work to axiomatize mistakes. Having a better axiomatization of mistakes can enable agents to react to mistakes in a more intuitive way.

One possibility for this is a classification of mistakes according to properties of the domain in which the agent operates and the capablities of the agent itself. Such a classification could make it easier to define the spectrum of mistakes and the capabilities agents need to handle them appropriately.

An aspect of the representation that has not been pinned down is that of actions and plans in the agent and how these representations will interact with whichever planner and plan execution architecture the logic is associated with. This work to some extent depends on the representations of the planner and the plan executor chosen. Some work along those lines has been done earlier [151]. The interaction between the logic and the procedural reasoner is not only a problem of having the right data representation, but also one of keeping multiple representations of the same data consistent. The logic will also need to have access to and be able to modify the structures of the plan execution system as necessary. A system like PRS (see above) seems attractive in that PRS has a database that can be used to influence the operation of the agent and this database could serve as a means to transfer information from the logic to the plan executor.

An aspect of mistakes that needs more work is their diagnosis. Up to now, we have been diagnosing mistakes based on the defaults and preferences used. This is adequate for the simpler cases of mistaken beliefs, but it is not clear it is sufficient in cases that there are more complex relations between the defaults and preferences and where these relations are time sensitive. In that case it seems that the simple procedures we use for finding mistakes will not be sufficient and we will need to adopt more general diagnosis procedures (see the diagnosis section of the test suite). An advantage of our logic is that it facilitates meta-reasoning, so it is capable of diagnosing itself.

Finally, the ideas presented will be validated to the extent that they improve the behavior of agents in some “real” domain. This leads to the problem of what domain to use to test our system. There are several criteria that the domain should satisfy. One of them is that it should not be a toy-domain but on the other hand, it should not be so complex as to be impossible to handle. It would further be useful if the complexity of the domain could be easily varied. It should be easy to run experiments in the domain and it should be interesting enough for people to work in it. A domain that seems ideal is that of computer games [1, 98, 97]. These are different from games like chess where one only needs reasoning–they require agents with a mixture of fast reactions and reasoning to solve problems and puzzles. Some current games, Quake and Half-Life, for instance, provide SDKs (software development kits) [176, 35] that allow one to program different environments and different monsters. The programmable environments allows one to change the complexity of the domain. The monsters in Half-Life, for instance seem to be simple plan execution systems [34] that can sense and act in the environment and have a “think” proedure that is called periodically. This provides an interesting environment to program and test our mistake handling algorithms as well as to explore the effectiveness of reasoning as opposed to acting reactively or procedurally.

10.6 Conclusion

We have sketched a rather detailed account of the sort of logic one would need to respond in a reasoned way to mistakes in the case the agent takes actions taken in the world. The realization that there has been a mistake in either the beliefs or some actions can enable the system to compute that other beliefs, intentions or actions are mistaken. This can be used to prevent inappropriate actions from being executed or can trigger a repair of the inappropriate actions already taken. The ability to notice and reason flexibly about mistakes in this way is a useful attribute for an agent to have if it is to operate in a dynamic world about which it has incomplete and uncertain information.

We may not yet be at the point where we could implement a system that behaves like the traffic controller mentioned in the beginning, but we think we are on the right track, and that we can implement a better monster.

Chapter 11

Conclusions and future work

We have shown how it is possible to build systems that detect their own failures and respond to them appropriately. This approach has been tested in some aspects of language processing and of commonsense reasoning. There is a need for AI systems to exhibit such a behavior if they are to act flexibly in the world. We believe that the approach we presented here will be an important part of the design of such systems.

The principal contributions of this work are:

- The design and implementation of Alma/Carne, a general-purpose active logic reasoner. Alma/Carne is time-situated, tolerates contradictions and provides facilities for meta-reasoning. Alma/Carne has been used in a number of applications. We believe that Alma provides the representational and inferential resources required for an agent to detect and handle its own mistakes.
- The design and implementation of an algorithm implemented in Alma that automatically detects and handles mistaken beliefs in a domain-independent way. This was successfully tested on a large subset of a test-suite of non-monotonic reasoning examples collected from the literature.
- Algorithms for and implementations of systems for detecting and responding to mistakes in natural language processing applications. These were used for computing presupposition projection and implicature cancellation in cases not properly handled by previous algorithms.

Future work

The more important problems to be worked on in the future include:

- Axiomatizing the properties of mistakes and their relations to the beliefs of an agent and to the state of the world.
- Allowing a variety of systems of inference rules to be used in Alma, not just resolution.
- Devising heuristic and meta-reasoning techniques to control inference in Alma to keep space and time usage reasonable.
- Extending the range of examples handled by the presupposition projection and implicature cancellation systems.
- Formalizing a semantics for the system that handles mistaken beliefs (chapters 7 and 8) and characterizing the behavior of the system with respect to the semantics.

- Implementing techniques derived from experiments on human non-monotonic reasoning to deal with mistakes and verifying the degree to which these model human behavior.
- Augmenting the test-suite for non-monotonic reasoning so that it tests a greater variety of behaviors of non-monotonic systems, including temporal ones.
- Refining the design of the agent that detects and handles mistaken beliefs, intentions and actions, implementing one such agent, and testing it in various domains including that of computer games.

Appendix A

Test suite

We now list the examples of non-monotonic reasoning collected from the literature. This is the beginning of a test suite for non-monotonic logic implementations. There are 12 categories of tests, each category requiring similar capabilities of the reasoner. Within each category though, some examples are simpler than others. The categories are: 1. Simple default application; 2. Multiple consistent defaults; 3. Multiple inconsistent defaults with explicit or implicit preferences; 4. Multiple inconsistent defaults with no preference between them; 5. the Closed World Assumption; 6. Epistemic reasoning; 7. Reasoning about names; 8. Reasoning about action; 9. Reasoning with assumptions; 10. Diagnostic reasoning; 11. Minimization; and 12. Miscellaneous problems.

The problem that were expressed in some specific formalism have all been translated to English sentences since we do not presuppose any formalism. We also have added sentences as necessary to make the examples complete.

A.1 Simple default application

These are basic examples of applying a single default and obtaining the conclusion of that default.

1. The prototypical application of non-monotonicity: *Birds typically fly*
   Tweety is a bird
   We conclude that Tweety flies

2. *Source:* [5]
   Illustration of prototypical reasoning.

   Typically children have (living)parents.

3. *Source:* [104]
   Example LB-A1[1]: basic default reasoning Assumptions:

   - Blocks A and B are heavy.
   - Heavy blocks are normally located on the table.
   - A is not on the table.

   Conclusions

[1]LB-XX refer to Lifschitz benchmark problems.

- B is on the table.

This illustrates that even though heavy blocks are usually on the table, we do not always conclude that. In this case we know non-defeasibly that A is not on the table, so the default does not apply in this case. In addition, the non-application of the default to A does not affect its application to B.

4. *Source:* [104]
Example LB-A2 illustrates that irrelevant information does not affect the application of the default.

   Assumptions:

   - Blocks A and B are heavy.
   - Heavy blocks are normally located on the table.
   - A is not on the table.
   - B is red.

   Conclusions

   - B is on the table.

   The fact that B is red does not affect its being on the table.

5. *Source:* [5]
If someone has a birthday, their friends give them gifts.
It is the birthday of Tweety.
Joe is Tweety's friend.
Does Joe give Tweety a gift?

6. *Source:* [5]
An illustration of "no-risk reasoning".

   In the absence of evidence to the contrary, assume that the accused is innocent
   If Tweety is accused, is he innocent?

7. *Source:* [5]
An example of "best guess reasoning".

   I know that there are some shopping centers in my city, and some of them are open on Sundays but I don't know which one is. On a Sunday I would simply drive to the closest one though I do not have any evidence that it will be open. It is simply the more convenient conjecture I can make.

8. *Source:* [157]
This illustrates expressing exceptions.

   Few Americas are socialists.
   Tweety is America-n.
   Is Tweety socialist?

9. *Source:* [157]
This illustrates handling incomplete knowledge.

   Whenever x is a person, then in the absence of information to the contrary, assume that home-town(x) = Palo Alto.
   Tweety is a person.
   Where does Tweety live?

10. *Source:* [5]
    This illustrates the use of non-monotonic logic in law.

    According to German law, a foreigner is usually expelled if they have committed a crime. One of the exceptions to this rule concerns political refugees. Tweety and Joe are foreigners and have committed a crime.
    Tweety is a political refugee.
    Is Tweety expelled?
    Is Joe expelled?

11. This is an extended example from [17]which uses epistemic concepts to do nonmonotonic reasoing. The epistemic ideas do not seem central to these examples so they were represented in our logic using the usual default approach.

    Germans typically drink beer is interpreted as:

    $$\forall x \; German(x) \wedge M \;\; DrinksBeer(x) \rightarrow DrinksBeer(x)$$

    With

    $$German(peter)$$

    We want to derive $DrinksBeer(Peter)$.

A.2 Multiple consistent defaults

In this case, there are several defaults in the logic that apply and are not inconsistent with each other. This does not cause any contradictions but the interactions (or lack of interaction) between the several defaults can be interesting.

1. *Source:* [104]
   Example LB-A3 illustrates that an object being an exception to one default does not make it exceptional for other defaults.

   Assumptions:

   - Blocks A and B are heavy.
   - Heavy blocks are normally located on the table.
   - Heavy blocks are normally red.
   - A is not on the table.
   - B is not red.

   Conclusions

   - B is on the table.
   - A is red.

2. *Source:* [5]

   Usually I go fishing on Sundays unless I wake up late. When I am on holidays I usually wake up late. Suppose it is Sunday and I am on holidays. We expect either default to apply.

3. *Source:* [5]
   This is an illustration of undesirable joint consistency.

   If I may assume that the weather will be bad, I'll take my sweater.

   If I may assume that the weather will be good then I'll take my swimsuit

   We should be cautious and take both. The desired answer seems to be derivable by assuming the premises of both defaults are true, but that is inconsistent.

4. *Source:* [161]
   Reiter and Criscuolo offer a set of examples that exemplify various interactions between defaults. Some of these cases involve inconsistency among the defaults, but we list all of them here to keep the examples together. In all these examples, we assume that *typically As are Bs*, and *typically Bs are Cs*. Each example adds more conditions:

   Typically As are Bs. Typically Bs are Cs. No A is a C

5. *Typically As are Bs. Typically Bs are Cs. All As are Cs*

6. *Typically As are Bs. Typically Bs are Cs. Typically As are Cs*

7. *Typically As are Bs. Typically Bs are Cs. It is not the case that As are typically Cs*

8. *Typically As are Bs. Typically Bs are Cs. Typically Bs are not As. It is not the case that As are typically Cs.*

9. *Typically As are Bs. Typically Bs are Cs. Typically As are not Cs*

10. *Typically As are Bs. Typically Bs are Cs. Typically Bs are not As. Typically As are not Cs.*

    Source: [161]
    This set of examples is similar to the above except that instead of assuming that "Typically As are Bs", we assume "All As are Bs". This will make a difference in cases where there is a conflict that involves that default.

11. *Assume: All As are Bs and typically Bs are Cs and No A is a C.*

12. *Assume: All As are Bs and typically Bs are Cs and All As are Cs.*

13. *Assume: All As are Bs and typically Bs are Cs and Typically As are Cs.*

14. *Assume: All As are Bs and typically Bs are Cs and It is not the case that As are typically Cs.*

15. *Assume: All As are Bs and typically Bs are Cs and Typically Bs are not As. It is not the case that As are typically Cs.*

16. *Assume: All As are Bs and typically Bs are Cs and Typically As are not Cs.*

17. *Assume: All As are Bs and typically Bs are Cs and Typically Bs are not As. typically As are not Cs.*

A.3 Multiple inconsistent defaults

In these cases, there are several defaults that could apply because their antecedents are true, but they cannot apply jointly. The main problem then is to have the right ones apply. There are two main classes of problems: those in which the preferences between the defaults are explicitly represented in the problem and those in which it is not. In the latter cases, the logics will have to infer the preferences or these could be added explicitly.

A.3.1 Explicit preferences

We first have some examples that can be resolved using a single preference between a pair of defaults.

1. *Source:* [17]
   This is a version of the Nixon diamond where a preference is stated between the defaults.

   Republicans are typically not pacifists.
   Quakers are usually pacifists.
   Nixon is a republican.
   Nixon is a quaker.
   We prefer the first default to the second.

2. *Source:* [17]

   Consider the following theory:
   Students are typically not married.
   Adults are typically married.
   Peter is a student.
   Peter is an adult.
   Is Peter married?

   In this case there is no preference and there is not way to choose. If we also say that *Adults that are not students are married* we should derive that Peter is not married.

3. *Source:* [17]
   A variation of the above is the following:
   Students are typically not married.
   Adults who are not students are typically married.
   Bearded people are typically students.
   Bearded people are typically adults.
   Peter is an bearded.

   We want to derive that Peter is not married.

   This time the facts that Peter is a student and an adult are obtained through defaults.

4. *Source:* [17]
   Here we have several levels of preference:

   (a) Tweety is a bird, Tweety is a penguin, Tim is a Penguin, Tim flies.

   (b) Penguins typically don't fly.

   (c) Birds typically fly.

   The higher the index, the higher the preference for the formula. We want to conclude that Tweety does not fly.

5. *Source:* [104]
   Benchmark LB-A4 involves "possible exceptions".

   Assumptions:

   - Blocks A and B are heavy.
   - Heavy blocks are normally located on the table.
   - A is a possible exception to this rule.

   Conclusions

   - B is on the table.

6. *Source:* [104]
   Benchmark LB-A9. There is a preference for Mary's testimony. Assumptions:

   - Jack asserts that block A is on the table.
   - Mary asserts that block A is not on the table.
   - When Jack asserts something he is normally right.
   - When Mary asserts something she is normally right.
   - Mary's evidence is more reliable than Jack's.

   Conclusions

   - Block A is not on the table,

7. *Source:* [104]
   Benchmark LB-A10 generalizes the default that Jack and Mary are normally right. Assumptions:

   - Jack asserts that block A is on the table.
   - Mary asserts that block A is not on the table.
   - When people assert something they are normally right.
   - Mary's evidence is more reliable than Jack's.

   Conclusions

   - Block A is not on the table,

8. *Source:* [17]
   Here the defaults at level T1 are to be preferred to those at T2 and those at T2 to those at T3.

   T1 *Hans attack Peter.*
   Peter injures Hans.

   T2 *If x attacks y, then y acts in self-defense.*
   If x acts in self-defense, x is not guilty.

   T3 *If x injures y, x is guilty*

   We want to conclude that Peter is not guilty.

9. *Source:* [5]
   This is a belief revision example with different degrees of entrenchment.

   Sicilians are normally hotheaded.
   Blonde persons are normally not hotheaded.

   Fiora, a sicialina is hotheaded, Johanna a blonde German is not and Rachel a blonde sicilian is.

   In this last set of examples, we have longer chains of preferences.

10. *Source:* [17]
    In this case, there are several preferences and these are conditional on other facts.

    Usually one has to go to a project meeting.
    This rule does not apply if one is sick, unless the sickness is only a cold.
    The rule is also not applicable if one is on vacation.

    Does Joe have to go to the meeting if

    - He is happy.
    - He is sick.
    - He has a cold.
    - He is on vacation.
    - He is on vacation and has a cold.

11. *Source:* [108]
    This extends the Tweety example with more defaults and preferences.

    Things normally don't fly.
    Birds are not normal things.
    Birds typically fly.
    Ostriches and penguins are abnormal birds.
    Ostriches and penguins typically don't fly.
    Ostriches, canaries and penguins are birds.
    Birds are typically feathered.
    The species are disjoint.

    Given that Tweety is a canary and Joe an ostrich, what can we conclude?

12. *Source:* [17]

    This example formalizes frame systems in a logic.

    (a) Cars typically have 4 wheels.
    (b) Cars typically have 5 seats.
    (c) Cars typically have 4 cylinders.
    (d) Sportscars typically have 2 seats.
    (e) Sportscars typically have 6 cylinders.
    (f) All sportscars are cars.
    (g) Speedy is a sportscar.

    We also have the following preferences:

    $$5 < 3, 4 < 2, \{6, 7\} < \{1, 2, 3, 4, 5\}$$

    We need to derive that Speedy has 2 seats, 6 cylinders and 4 wheels.

A.3.2 Multiple inconsistent defaults with implicit preferences

In this group of examples, the preferences are not explicitly stated in the problem. The reasoner should be able to derive those. These are essentially the same as the above set except that the preferences are not explicit here.

1. Source: [5]

   Bill is a high school dropuot. Tyupically high school dropouts are adults. Typically adults are employed

2. *Source:* [17]

   The SPD believes that people should work less but not for less money.
   The LAF, a subgroup of the SPD believes that we need to work less, and get less money.
   Company owners think that people should work more for less money.

   What are the opinions of someone who is

   - In the SPD.
   - In the LAF.
   - A company owner.
   - A company owner in the SPD.
   - A company owner in the LAF.

3. *Source:* [54]

   Molluscs are normally shell bearers.
   Cephalopods must be Molluscs but normally are not shell bearers.
   Nautili must be Cephalopods and shell-bearers.

   We should be able to derive that

   - Given a nautilus, it is a cephalopod, a mollusc, and a shell-bearer.
   - A cephalopod not known to be a nautilus is a mollusc with no shell.

4. *Source:* [183]
   Elephants are typically gray.
   Royal elephants are elephants but are not typically gray.
   Circus elephants are royal elephants.
   Clyde is a circus elephant.

   Is Clyde gray? What if we add the statement that Clyde is an elephant?

A.4 Multiple inconsistent defaults with no preference

In this set of examples, we have a set of defaults that have inconsistent consequences and there is no implicit or explicit preference among any of the defaults.

1. *Source:* [161]
   The Nixon diamond is perhaps the simplest sucha case. *Typically republicans are not pacifists.*
   Typically quakers are pacifists.
   John is both a quaker and a republican. Is John a pacifist?

2. *Source:* [183]
   In this variant there is an extra inference one must make to conclude that Nixon is not pacifist.

   Quakers are typically pacifists.
   Pro-defense people are typically not pacifists.
   Republicans are typically pro-defense.
   Nixon is both a quaker and a republican.

   Is Nixon a pacifist?

3. *Source:* [157]

   Assume a person's hometown is that of his/her spouse.
   Assume a person's hometown is where his/her employer is located.
   Mary's spouse lives in Toronto, her employer is in Vancouver.

   Where does Mary live?

4. *Source:* [17]

   Dogs or birds are pets.
   Dogs aren't birds.
   Pets are usually dogs.
   Singing things are usually birds.

   Given that Joe sings, we want to conclude that it is a bird and a pet. The answer should not change if we also assert that Joe is a pet.

5. *Source:* [113]
   The lottery paradox is the same kind of problem: there are a number of defaults that are inconsistent with other knowledge.

   A lottery is held with 1 million participants, including our friend John, The odds of John's winning are so low that we infer by default that he won't. Yet by the same token we can infer that none of the other 999,999 members will win, which contradicts our knowledge that at least one person must win.

A.5 Closed World Assumption

The example here illustrates the closed world assumption.

Source: [156]

There are 4 teachers: a, b, c, d.
There are 3 students: A, B, C.
a teaches A, b teaches B, c teaches C and a teaches B.

Who does not teach B?

A.6 Epistemic reasoning

Epistemic and auto-epistemic reasoning have been used as one way to formalize non-monotonic logics. In this section we list examples that use knowledge about what the agent knows explicitly. Examples in the literature that are formalized in auto-epistemic logic but could be equally formalized in a different way are not presented.

1. *Source:* [5]

   Are the Rolling Stones giving a concert in Newcastle next week?
   No, because otherwise I would have heard about it.

   If we later buy the newspaper and find: "The concert of the century: The Rolling stones in Newcastle next week!"*. The conclusion should change.*

2. *Source:* [104]
   Example LB-E1

   Assumptions:

   - Block A is on the table

   Conclusions

   - It is not known whether B is on the table.

3. *Source:* [104]
   Example LB-E2

   Assumptions:

   - at least one of the blocks A, B, is on the table.

   Conclusions

   - It is not known whether A is on the table.
   - It is not known whether B is on the table.

4. *Source:* [104]
   Example LB-E4

   Assumptions:

   - Blocks that are not known to be heavy are on the table.
   - Block A is heavy.

   Conclusions

   - Block B is on the table.

A.7 Reasoning about names

This set of examples involves issues surrounding uniqueness of names. Once again the usual nonmonotonic reasoning mechanisms are used but in this case, a distinction has to be made between names and the objects they name.

1. *Source:* [104]
   Example LB-C1

   Assumptions:

- Different names normally denote different objects
- The names “Ray” and “Reiter” denote the same person.
- The names “Drew” and “McDermott” denote the same person.

Conclusions

- The names “Ray” and “Drew” denote different people.

2. *Source:* [104]
   Example LB-C2

   Assumptions:

   - Different people normally have different fathers.
   - Joseph and Benjamin have the same father.
   - Gaius and Tiberius have the same father.

   Conclusions

   - Joseph and Gaius have different fathers.

3. *Source:* [52]
   Tom’s telephone number is the same as Sue’s.
   Bill’s telephone number is 555-1234.
   Mary’s telephone number is not 555-1234.
   All the people are different form each other.

   Is Tom’s telephone number 555-1234? Is Tom’s telephone number the same as Mary’s?

A.8 Reasoning about action

There are several such examples in the literature, represented in different formalisms, mainly situation based or interval based. The same nonmonotonic behavior is used in both sorts of examples. Our logic gives the results expected.

1. The following examples are from [104]

   Assumptions:

   - After an action is performed things normally remain as they are.
   - Any time a robot grasps a block, the block will be in the hand.
   - If a block is in the hand then, after the robot moves it to the table, the block will be on the table.
   - Initially block A is not in the hand.
   - Initially block A is not on the table.

   Conclusions

   - After the robot grasps block A, waits, and then moves it onto the table, the block will be on the table.

2. In this example, the representation involves exceptions to the action specifications.

   Assumptions:

- After an action is performed things normally remain as they are.
- When the robot grasps a block, the block will be in the hand.
- When the robot moves a block onto the table, the block will normally be on the table.
- Moving a block that is not in the hand is an exception to this rule.
- Initially block A is not in the hand.
- Initially block A is not on the table.

Conclusions

- After the robot grasps block A, waits, and then moves it onto the table, the block will be on the table.

3. This example seems to be problematic since it states that a block is on the table iff it is not on the floor. However, the block can also be in the hand.

   - After an action is performed, things normally remain as they are.
   - A block is on the table, if and only if it is not on the floor.
   - When the robot grasps a block, the block will be normally in the hand.
   - When the robot moves a block onto the table, the block will normally be on the table.
   - Moving a block that is not in the hand is an exception to this rule.
   - Initially block A is not in the hand.
   - Initially block A is on the floor.

   Conclusions

   - After the robot grasps block A, waits, and then moves it onto the table, the block will not be on the floor.

   In these examples from [168]. The representation in this case is interval rather than situation based.

4. **The Yale shooting scenario.**

   The world consists of:

   - Two truth valued features specifying whether the gun is loaded and whether the bird is alive.
   - Actions: load gun, fire gun, wait.

   The effect of loading is that the gun is loaded. If gun is loaded at beginning of firing, then bird is dead at the end of firing and the gun is unloaded, otherwise firing has no effect.

   Initially, the bird alive and the gun not loaded.

   Events: load gun, wait, fire.

   Conclusion: the bird is not alive.

5. **The Stockholm delivery scenario.**

   Consider a box, B, and a car, C, both of which are located in the city of Linkoping at time 0, which represents the beginning of the scenario. The box is not in the car at time 0. Two action types are considered, namely to load a box into the trunk of a car, and to drive a car to a specified city. From time 8:15 to time 8:20 the box is loaded into the car; from time 8:40 to time 11:15 the car is driven to Stockholm.

   Question: where is the box at time 13:00?

6. **The rainy day shooting scenario.**

   Consider a world similar to the Yale shooting scenario with three propositional features: a *('Fred is alive'),* l *('The gun is loaded'),* p *('It is raining'). There is one action* Fire *whose effect is to unload the gun if it was loaded and of killing Fred iff the gun was loaded. The rain does not affect these outcomes, and the rain itself is not affected. Assume that in the case that Fred is alive and the gun is loaded, the action takes two time units, where the gun becomes unloaded in the first and Fred dies in the second timestep. In other cases it takes just one timestep.*

   If it is raining and the gun is loaded and fires, does Fred die?

7. *Source:* [108]

   Unless something prevents it, x is on y in the situation that results from the action move(x, y).
   If either x or y is not a block, that prevents the move.
   If x or y is not clear, that prevents a move.
   If y is too heavy, that prevents a move.

   Given blocks A and C, we want to conclude that we can move A to C in situation S0.

A.9 Reasoning with assumptions

There are a number of examples in the literature where one has to assume a fact and reason based on this assumption. The assumption may not always be explicit but the structure of the problem seems to require that.

1. *Source:* [104]
   We need to assume that Mary's evidence is more reliable to come to the conclusion. Assumptions:

   - Jack asserts that block A is on the table.
   - Mary asserts that block A is not on the table.
   - When people assert something they are normally right.

   Conclusions

   - If Mary's evidence is more reliable than Jack's then block A is not on the table.

2. *Source:* [168]
   The hiding turkey scenario This involves making assumptions in the context of reasoning with action.

   The world is as the Yale Shooting Scenario with two additional fluents: deaf turkey; hiding turkey. If the turkey is not deaf, then when gun is loaded it goes into hiding. Firing only kills turkey if it is not hiding.

   Initially the turkey is alive, not hiding, and the gun is not loaded. It is unknown if turkey is deaf.

   The events are: load gun, wait then fire.

   The conclusion we want to reach is that either the turkey is deaf and dead or non-deaf and alive.

3. *Source:* [168]
   The ferryboat connection scenario

   The world consists of a motorcycle having positions: "On Island Fyen", "At ferryboat landing", "On ferry-boat", "In Jutland".
   Initially motorcycle is driving along a road on island Fyen, in direction of the ferryboat landing. The ferryboat departs at 23:01. the last time for arrival is 23:00. If motorcycle is on board at 23:01, it will arrive in Jutland

between 23:45 and 23:50 otherwise it stays. The motorcycle reaches the landing sometime between 22:55 and 23:05.

Conclusion: at 23:55, motorcycle is either at the landing or in Jutland.

4. *Source:* [168]

   The Russia turkey scenario

   The world is the same as in the Yale Shooting Scenario with an additional action: spinning the gun chamber. The effect is that the gun may be randomly unloaded after that action.

   Initially the gun is unloaded, the turkey is alive.

   The events are: load the gun, spin the chamber, then fire.

   There are two possible conclusions:either the gun gets unloaded and there is no change after firing or there is no change from spinning, and the turkey is dead.

5. *Source:* [168]
   The furniture assembly scenario

   The world consists of a furniture kit with two features: whether it is assembled and whether if contains instructions. The action is assembling the kit. If kit was unassembled this takes 20 minutes if instructions included, otherwise it takes 60 minutes.

   Initially the kit is unassembled and it is not known whether the instructions are included.

   There are two possible conclusions: the instructions are included and the kit is assembled in 20 minutes, or the instructions are not included and kit is assembled in 60 minutes.

6. *Source:* [17]
   This is a reasoning by cases example: *Italians like wine.*
   Frenchmen like wine.
   John is Italian or John is a Frenchman.

   We want to conclude that John likes wine.

A.10 Diagnosis/explanation reasoning

Several examples of nonmonotonic reasoning involve diagnosing problems or require explanations to be inferred for some events.

1. *Source:* [104]
   Example LB-D4. This requires the reasoner to infer the conditions at the beginning of the sequence of actions given the outcomes. Assumptions:

   - After an action is performed, things normally remain as they are.
   - When the robot grasps a block, the block will normally be in the hand.
   - When the robot moves a block onto the table, the block will normally be on the table.
   - Moving a block that is not in the hand is an exception to this rule.
   - Initially block A was not on the table.
   - After the robot moves A to the table and then waited, A was on the table.

   Conclusions

- Initially A was in the hand.

2. *Source:* [104]
   Example LB-D5. Assumptions:

   - After an action is performed, things normally remain as they are.
   - When the robot grasps a block, the block will normally be in the hand.
   - When the robot moves a block onto the table, the block will normally be on the table.
   - Moving a block that is not in the hand is an exception to this rule.
   - Initially block A was not on the table.
   - Initially block A was not in the hand.
   - After the robot grasped some block and then moves some block onto the table, A was on the table.

   Conclusions

   - The block that was grasped was A
   - The block that was moved to the table was A.

3. *Source:* [104]
   Example LB-D6. Assumptions:

   - After an action is performed, things normally remain as they are.
   - When the robot grasps a block, the block will normally be in the hand.
   - When the robot moves a block onto the table, the block will normally be on the table.
   - Moving a block that is not in the hand is an exception to this rule.
   - Initially block A was not on the table.
   - Initially block A was not in the hand.
   - After the robot performed 2 actions. A was on the table.

   Conclusions

   - The first of the two actions was grasping A.
   - The second of the two actions was moving A to the table.

4. *Source:* [104]
   Example LB-D8. Assumptions:

   - After an action is performed, things normally remain as they are.
   - When the robot moves a block to another location, the block will be normally at that location.
   - After the robot moved a block to Location 1 and then to Locations2, the block changed its color.

   Conclusions

   - The block changes its color only once, either after the first move or after the second.

5. *Source:* [104]
   Example LB-D9. Assumptions:

   - When the robot moves a block to another location, the block will be normally at hat location.
   - After the robot moved block A onto the table, and then moved block B onto the table, at most one of the blocks a, B was on the table.

Conclusions

- After the two actions were performed, exactly one of the blocks A, B was on the table.

6. *Source:* [104]
   Example LB-D10 Assumptions:

   - After an action is performed, things normally remain as they are.
   - When the robot moves a block to another location, the block will be normally at that location.
   - After the robot moves block A to location 1, block B change color.

   Conclusions

   - Block B would have changed its color if the robot had moved A to location 2.

7. *Source:* [104]
   Example LB-D11 Assumptions:

   - When two actions are performed concurrently, their effects are normally combines.
   - After a block is moved to another location, it is normally at that location.

   Conclusions

   - After block A is moved to Location 1 and block B is concurrently moved to location2, A will be at location 1 and B will be at location 2.

8. *Source:* [168]
   The Stanford murder mystery.

   The world is the same as that for the Yale shooting scenario. The turkey is initially alive and the events are to fire the gun and to wait. The result is that the turkey is dead at the end of the firing. We want to conclude that the gun was initially loaded and that the turkey was not alive at the end of the firing.

A.11 Minimization

These examples have to do with inferring an explicit minimization of a predicate. There are two sorts of minimization here: minimizing by instances and minimizing by subsets. In the first case, we know of some objects that don't satisfy a default and in the second, we know of subsets of a set of objects that don't satisfy the default. The inference desired is to qualify the default with these exceptions. We can also have combinations of these two cases.

This sort of reasoning is commonly found in circumscription where one minimizes predicates before using the minimizes predicates to make inferences.

Subsets

1. *Source:* [104]
   Assumptions:

   - Animals normally do not fly.
   - Birds are animals.
   - Birds normally fly.
   - Ostriches are birds.

- Ostriches normally do not fly.

Conclusions

- Animals other than birds do not fly.
- Birds other than ostriches fly.
- Ostriches do not fly.

2. *Source:* [104]
   Assumptions:

   - Animals normally do not fly.
   - Birds are animals.
   - Birds normally fly.
   - Bats are animals.
   - Bats normally fly.
   - Ostriches are birds.
   - Ostriches normally do not fly.

   Conclusions

   - Animals other than birds and bats do not fly.
   - Birds other than ostriches fly.
   - Bats fly.
   - Ostriches do not fly.

3. *Source:* [104]
   This example is similar to the above. In this case, the exceptions are specified by defaults.

   Assumptions:

   - Quakers are normally pacifists.
   - Republicans are normally not pacifists.

   Conclusions

   - Quakers who are not republicans are pacifists.
   - Republicans who are not Quakers are not pacifists.

4. *Source:* [104]
   From the same set of examples, this time there is a property (politically active) for which transitivity must apply. However this is not something that we would like to do generally.

   Assumptions:

   - Quakers are normally pacifists.
   - Republicans are normally hawks.
   - Pacifists are normally politically active.
   - Hawks are normally politically active.
   - Pacifists are not hawks.

   Conclusions

- Quakers who are not republicans are pacifists.
- Republicans who are not Quakers are hawks.
- Quakers, Republicans, Pacifists, Hawks are politically active.

5. In the following cases we consider minimization in terms of instances.
   Source: [104]
   Assumptions:
   - Blocks A and B are heavy.
   - Heavy blocks are normally located on the table.
   - A is not on the table

   Conclusions
   - All heavy blocks other than A are on the table.

6. *Source:* [104]
   Assumptions:
   - Blocks A and B and C are heavy.
   - Heavy blocks are normally located on the table.
   - At least one of A, B is not on the table

   Conclusions
   - C is not on the table.
   - Exactly one of A, B is not on the table.

   In this case we need to specify that *exactly one* block does not satisfy the default.

7. *Source:* [104]
   Assumptions:
   - Heavy blocks are normally located on the table.
   - At least one heavy block is not on the table.

   Conclusions
   - Exactly one heavy block is not on the table.

8. *Source:* [104]
   Assumptions:
   - Block A is on the table.

   Conclusions
   - About any block other than A it is not known whether it is on the table.

9. *Source:* [108]
   In this circumscription we want to minimize the extent of $block$ as above:
   If we know that A, B and C are blocks, we want to conclude that the only blocks are A, B and C.

10. *Source:* [108]

    If we know that either A or B is a block, we want to conclude that the only block is A or the only block is B.

A.12 Miscellaneous

The problems in this category include reasoning with inconsistent justifications, reasoning with counting, and reasoning about integers that don't fit well in the previous categories.

In this set we collect examples that do not fit well in any of the other sets.

1. *Source:* [5]

   This example is about the joint consistency of justifications.

   By default a robot's arm (a or b) is usable unless it is broken; further we know that either a or b is broken. Given this information, we do not expect both a and b to be usable. (Poole)

2. *Source:* [104]

   Assumptions:

   - Block A is heavy
   - Heavy blocks are normally located on the table.
   - At least one heavy block is not on the table.

   Conclusions

   - A is on the table.

3. *Source:* [104]
   Assumptions:

   - After an action is performed, things normally remain as they are.
   - When a person accepts a job offer from some employer, he will be employed by that employer.
   - If Bill is offered a job at Berkeley or at Stanford when he is unemployed, he will accept it.
   - Bill is currently unemployed.

   Conclusions

   - After Bill is offered jobs at Berkeley and Stanford at two different instants of time, he will be employed either by Berkeley or by Stanford.

4. *Source:* [52]
   This is an example of reasoning with inconsistency.

   $$\exists x\ N(x) \wedge \forall y\ (N(y) \rightarrow x \neq succ(y))$$
   $$\forall x\ N(x) \rightarrow N(succ(x))$$
   $$\forall xy\ succ(x) = succ(y) \rightarrow x = y$$

   If we circumscribe N we get a contradiction.

5. In [145] we have this reasoning episode:

   > Tweety is a bird; so (I may as well assume) Tweety can fly; but Tweety is an Ostrich and so cannot fly after all; my belief that Tweety could fly was false, and arose from my acceptance of a plausible hypothesis. Do all birds that may fly (as far as I know) in fact fly? No, that's just a convenient rule of thumb.

Appendix B

Test results

We tested our non-monotonic logic on the test suite presented in appendix 11. We illustrated in detail an instance of the execution of our logic with a problem of each class of the test suite in chapter 9. We here give a fuller account of the problems that were solved and those that were not and some brief comments on intereting problems. The axiomatization of these problems are available at our website. Also available there is the possibility to "replay" our solutions to these problems. Note that the problems are identified by their index in the test suite.

The logic we have described earlier solves a wide range of problems in the literature in the expected way. Most of the problems it does not solve belong to two categories that would need more facilitied in the logic or in Alma before they can be solved. Their solution is not precluded by the logic–it is only a matter of more work.

B.1 Simple default application

Problems A.1.1–A.1.4 were straightforwardly solved in a way similar to A.1.3. The other problems in this category are similar to these and were not attempted.

B.2 Multiple consistent defaults

Problems A.2.1–A.2.10 were solved. Problems A.2.11–A.2.17 were similar to and simpler than A.2.4–A.2.10 and were not attempted.

B.2.1 Problem A.2.3

We have three axiomatizations of A.2.3. In the first representation of the problem, we introspect to see whether we know that the weather is bad or not. If we don't, we take both sweater and swimsuit. If we do know that the weather is bad though, we take the sweater and if it is good, we take the swimsuit. However this solution tests for the three cases of not knowing anything, knowing the weather is bad and knowing that the weather is not bad separately and has an explicit solution for each case. This is not desirable.

In the second representation, we take both the sweater and the swimsuit by default, but if the weather is bad, we don't take the swimsuit, and if the weather is good, we don't take the sweater. In this case, not knowing anything about the weather intially leads to taking both sweater and swimsuit. But then, these imply that it the weather is both good and not good which results in a contradiction and we take neither.

In the third representation, we interpret the “if I may assume P” clause by the failure to introspect the negation of P. If we don’t introspect that the weather is not bad, we can assume that the weather is bad; and if we fail to introspect that the weather is not bad, we assume it is not bad. But if we know that the weather is bad, we cannot assume that it is not bad. Similarly for the weather being not bad. From these explicit assumptions, we infer what to take. This representation works as intended: if we don’t know whether the weather is bad or not, we take both sweater and swimsuit. But if we then learn (or learn initially) that the weather is bad, we don’t take the swimsuit.

B.2.2 Problem A.2.7

In this problem we want to deny that As are typically Cs, which is different from asserting that As are typically not Cs. The difference is apparent if we have an object that is an A but not a B. In the first case, there is no conclusion whereas in the second case we assert that the object is not a C.

The denial is represented by a default that has as antecedent the conjunction of the conditions under which we usually derive that an object which is an A is a C. The consequent of the default then asserts that that the object is not a C. This causes a contradiction which results in suspending belief about the object being a C. Since the antecedent of this denying default will only be derivable when the fact that the object is a C is derived, we do not get the problem of asserting spurious negations as above. A shortcoming of this solution is that it needs us to know all the ways in which one can go from knowing an object is an A to knowing it is a C.

B.3 Multiple inconsistent defaults

All the problems in this section were solved, except for A.3.A.3.1.6 and A.3.A.3.1.6 which were similar to the others and not attempted.

B.3.1 Multiple inconsistent defaults with implicit preferences

All the problems here, except for A.3.**??**.4 were solved. That problem was similar to the others and not attempted.

B.4 Multiple inconsistent defaults with no preference

All the problems in this section, except for A.4.3 were correctly solved. A.4.3 involves two defaults, the first that a person lives where her employer is and second that a person lives where her spouse lives. In the case of Mary, her employer and spouse are in different cities. The desired result is that Mary lives either in the employer’s or the spouse’s town. Our logic does not conclude anything about where Mary lives (we also state that a person can’t live in two places).

B.5 Closed World Assumption

The only problem here was solved in the expected way.

B.6 Epistemic reasoning

All the problems in the A.6 category were solved using Alma's introspection ability.

B.7 Reasoning about names

This set of examples involves issues surrounding uniqueness of names. Once again the usual nonmonotonic reasoning mechanisms are used but in this case, a distinction has to be made between names and the objects they name. All but A.7.3 were solved. That was similar to the previous ones and was not attempted.

B.8 Reasoning about action

All the examples in this section were solved except for A.8.**??** which is similar to A.8.4. Our approach worked equally well for the situation and the event based representations.

B.9 Reasoning with assumptions

Alma does not support reasoning with assumptions, therefore none of the problems in this section were solved. These could be attempted after that sort of reasoning is added to Alma.

B.10 Diagnosis/explanation reasoning

Our non-monotonic logic and Alma perform a very simple form of diagnosis to find the causes of a contradiction. This is not sufficient for the usual problems in diagnosis encountered in the literature. These problems could be solved in our framework using the metalogical prediated and operatores available. However, the algorithms for doing that need to be implemented.

B.11 Minimization

These examples have to do with inferring an explicit minimization of a predicate. There are two sorts of minimization here: minimizing by instances and minimizing by subsets. In the first case, we know of some objects that don't satisfy a default and in the second, we know of subsets of a set of objects that don't satisfy the default. The inference desired is to qualify the default with these exceptions. We can also have combinations of these two cases.

This sort of reasoning is commonly found in circumscription where one minimizes predicates before using the minimizes predicates to make inferences.

An example of the first kind: if birds usually fly, but we know that the birds anna and bob don't, then we infer that *all* birds fly except if they are anna or bob. In the second case, if we know that usually birds fly but that sick birds and penguins don't, then we infer that *all* birds fly except for penguins and sick birds.

This is not a sort of inference that we would generally like to make because typically, we cannot know when we have found all the exceptions and the universally quantified formula we infer is likely to be false. However, in cases that we do think we know all the exceptions, we can make that jump.

Note that this involves explicitly asserting that all odjects except for the exceptions satisfy the default. Without this assertion, a nonmonotonic logic should be able to infer every instance of the assertion by simply using its nonmonotonic inference. Here though, we want to assert that fact explicitly.

This sort of inference requires that we reason not with, but about the defaults. It involves finding *all* the exceptions to the defaults (we know of), then qualifying the defaults with these exceptions.

In parallel with the two kinds of minimization mentioned above, we have two kinds of exceptions: 1. instance exceptions which are objects that satisfy the antecedent of the default but not the consequent 2. subset exceptions which are identified by formulas that imply the antecedent and do not satisfy the consequent. In the example above, knowing that $Bird(Anna) \wedge \neg Flies(Anna)$ allows us to conclude that $Anna$ is an exception to the birds flying default (the same applies for Bob). Similarly, $Penguin(x) \hookrightarrow \neg Flies(x)$ and $Penguin(X) \rightarrow Bird(X)$ is sufficient for us to conclude that penguins are an exception to the default (the same applies for sick birds).

Once we have obtained the set of exceptions, we use them together with the default to generate the quantified generalization. Continuing the example, if the default is $Bird(X) \hookrightarrow Flies(X)$, we should get in the first case: $Bird(X) \wedge X \neq Anna \wedge X \neq Bob \rightarrow Flies(X)$. In the second case: $Bird(X) \wedge \neg Penguin(X) \wedge \neg Sick(X) \rightarrow Flies(X)$.

Implementing such an inference is possible in Alma since we can reason about the formulas in the knowledge base. However doing so it not straightforward as it involves several steps of computation to first of all detect and gather all the exceptions and secondly to take the default apart to assemble the qualified generalization. We have implemented this only for the case of exceptional subsets. A lot of the computation is done in the prolog helper programs.

To generate the qualified generalization, we first request that Alma find the exceptions to the defaults, and once this is done, we generate the generalization. Note that he same mechanism of finding the exceptions can be used to find specificity preferences.

A.11.1 and A.11.2 were solved. A.11.3 and A.11.4 are somilar to these and were not attempted. The other examples are minimization by instances. More prolog code needs to be written to solve these. This will be similar to the code written for the subset minimization. Once this is done, these problems too, are expected to be solved.

B.12 Miscellaneous

A.12.1 was solved as our logic is sensitive to any inconsistency, whether it is in the antecedents or in the conclusions. In this case, since neither $broken(a)$ nor $broken(b)$ can be derived, neither conclusion is asserted. The issue of consistency of the antecedents does not come into the picture here. A.12.2 was not solved because of problems with axiomatizing “at least” in our logic. A.12.4 was not solved as it explicitly required circumscription and while the reasoning A.12.**??** can be partially reproduced, not all of it can since that required reasoning about the defaults which this logic does not do, although that is possible in Alma.

Appendix C

Alma/Carne manual

Active logics [46, 48] have the following characteristics:

- they are situated in time
- they maintain a history
- they tolerate contradictions
- they enable meta-reasoning to be done

These characteristics make active logics suitable for use in various domains including time situated planning and execution [133, 151], reasoning about other agents' reasoning [47], discourse context updating [76], computation of Gricean implicatures [141], representation of meta and mixed-intiative dialog [144, 4].

For each of the domains above, it has typically been the case that when implemented, a new special purpose active logic had to be programmed. This document describes the implementation of a general-purpose active logic engine called Alma (Active Logic MAchine). The aim is that it should be possible to specify and execute any active logic using the language of Alma.

It is usually not sufficient to compute consequences of an active logic, we want the execution of the logic to be sensitive to the outside world and to have the logic affect the outside world. This is made possible by a procedure execution system called Carne that runs in conjunction with Alma. This document describes the Alma-Carne system.

After an overview of the Alma-Carne system, details of Alma are presented. First a description of Alma, then the method of running Alma and finally an account of the implementation. The same information is then presented for Carne. The next section describes some applications of the Alma-Carne system. The last two sections discuss the current bugs and the future development of the system. The appendices include a keyword reference and illustrations of running Alma-Carne.

The latest version of the complete manual can be found at http://www.cs.umd.edu/projects/active/Alma/manual.html.

Bibliography

[1] P. E. Agre and D. Chapman. Pengi: an implementation of a theory of activity. In *Proceedings of AAAI-87*, 1987.

[2] J. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[3] James F. Allen and George Ferguson. Actions and events in interval temporal logic. *Journal of Logic and Computation*, 4(5), 1994.

[4] C. Andersen, D. Traum, K. Purang, D. Purushothaman, and D. Perlis. Mixed initiative dialogue and intelligence via active logic,. In *Proceedings of the AAAI'99 Workshop on Mixed-Initiative Intell igence*, 1999.

[5] G. Antoniou. *Nonmonotonic reasoning*. MIT Press, 1997.

[6] Arruda, N. C. A. da Costa, and Chuaqui. *Mathematical logic in latin america*, chapter A survey of paraconsistent logic. North Holland, 1980.

[7] E. M. Atkins, E. H. Durfee, and K. G. Shin. Detecting and reacting to unplanned-for world states. In *Proceedings of AAAI-97*, pages 571–576, 1997.

[8] A. D. Baddeley. *Human memory: theory and practice*. Allyn & Bacon, 1990.

[9] Afzal Ballim and Yorick Wilks. *Artificial Believers: The ascription of Belief*. Lawrence Erlbaum Associates, 1991.

[10] A. J. Barnett. How much is control knowledge worth? a primitive example. *Artificial Intelligence*, 22(1):77–89, 1984.

[11] H. Barringer, M. Fisher, D. Gabbay, and A. Hunter. Meta-reasoning in executable temporal logic". In *KR'91: Principles of Knowledge Representation and Reasoning*, pages 40–49, 1991.

[12] D. Benanav. Recognizing unnecessary inference. In *Proc. of the 11th IJCAI*, pages 366–371, Detroit, MI, 1989.

[13] Salem Benferhat, Claudette Cayrol, Didier Dubois, Jérôme Lang, and Henri Prade. Inconsistency management and prioritized syntax-based entailment. In Ruzena Bajcsy, editor, *Proceedings of the 13th I.J.C.A.I.*, pages 640–645, Chambéry, Savoie, France, August 1993. Morgan Kaufmann.

[14] J.F.A.K van Benthem. *The Logic of time*. D. Reidel Publishing Company, Dordrecht, 1983.

[15] H. A. Blair and V. S. Subrahmanian. Paraconsistent logic programming. *Theoretical Computer Science*, 68(2):135–154, October 1989.

[16] M. Boddy and T. Dean. Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67(2):245–285, 1994.

[17] G. Brewka. *Nonmonotonic reasoning: logical foundations of commonsense.* Cambridge University Press, 1991.

[18] G. Brewka, J. Dix, and K. Konologe. *Nonmonotonic reasoning: an overview*. CSLI publications, 1997.

[19] R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47(1-3):139–159, January 1991.

[20] D. C. Brown and B. Chandrasekaran. *Design Problem Solving: Knowledge Structures and Control Strategies*. Morgan Kaufmann, Los Altos, CA, 1986.

[21] A. Bundy and B. Wellham. Using meta-level inference for selective application of multiple rewrite rule sets in algebraic manipulation. *Artificial intelligence*, 16(2):189–212, 1981.

[22] M. Cadoli and M. Lenzerini. The complexity of propositional closed world reasoning and circumscription. *Journal of computer and system sciences*, 48:255–310, 1994.

[23] M. Cadoli and M. Schaerf. Approximate entailment. In *Proc. of the second conference of the italian association of rartificial intelligence*, number 549 in LNAI, pages 68–77, 1991.

[24] M. Cadoli and M. Schaerf. A survey on complexity results for nonmonotonic logics. *Journal of Logic Programming*, 17(2–4):127–160, 1993.

[25] M. Cadoli and M. Schaerf. Approximate inference in default logic and circumscription. *Fundamentae Informaticae*, 23(1):123–143, 1995.

[26] M. Canale and M. Swain. Theoretical bases of communicative approaches to second language teaching and testing. *Applied Linguistics*, 1(1), 1980.

[27] Michael Canale. From communicative competence to communicative language pedagogy. In Jack C. Richards and Richard W. Schmidt, editors, *Language and communication*. Longman, 1983.

[28] C.-L. Chang and R. C.-T. Lee. *Symbolic logic and mathematical theorem proving*. Academic Press, 1973.

[29] M. Chapman. Everyday reasoning and the revision of belief. In *Mechanisms of everyday cognition*. 12th Biennial conference on lifespan developmental psychology, L. Erlbaum Associates, 1993.

[30] N. Chomsky. *Aspects of the theory of syntax*. MIT Press, 1965.

[31] William J. Clancey. The advantages of abstract control knowledge in expert system design. In *Proc. AAAI-83*, pages 74–78, Washington, D.C., August 1983.

[32] E. V. Clark. Awareness of language: some evidence from what children say and do. In A. Sinclair, R. J. Jarvella, and W. J. M. Levelt, editors, *The child's conception of language*. Springer-Verlag, 1978.

[33] Luca Console, Daniele Theseider Dupre, and Pietro Torasso. On the relationship between abduction and deduction. *Journal of Logic and Computation*, 1(5):661–690, 1991.

[34] S. Cook. Advanced: Understanging the monster ai. http://www.planethalflife.com/hlsdk2/sdk/understanding_the_monster_ai.htm.

[35] S. Cook. Introduction to half-life sdk2. http://www.planethalflife.com/hlsdk2/, 2000.

[36] R. Davis. Meta-rules: reasoning about control. *Artificial Intelligence*, 15:179–222, 1980.

[37] T. Dean. Decision-theoretic control of inference for time-critical applications. *International Journal of Intelligent Systems*, 6:417–441, 1991.

[38] K. Decker, K. Sycara, and M. Williamson. Middle-agents for the internet. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, Nagoya, Japan, 1997.

[39] A. del Val. Non-monotonic reasoning and belief revision: Syntactic, semantic, foundational and coherence approaches. *Journal of Applied Non-Classical Logics*, 7 (1-2):213–240, 1997.

[40] J. P. Delgrande. A first-order conditional logic for prototypical properties. *Artificial Intelligence*, 33(1):105–130, 1987.

[41] J. P. Delgrande. An approach to default reasoning based on first-order conditional logic: Revised report. *Artificial Intelligence*, 36(1):63–90, 1988.

[42] D. Denett. *The intentional stance*. MIT Press, 1987.

[43] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12(3):231–272, 1979.

[44] B. Drabble and A. Tate. O-plan: A situated planning agent. In *Proceedings of the Third European Workshop on Planning Systems*, 1995.

[45] D. Draper, S. Hanks, and D. Weld. A probabilistic model of action for least-commitment planning with information gathering. In *Proceedings of UAI94*, 1994.

[46] J. Drapkin and D. Perlis. A preliminary excursion into step-logics. In *Proceedings SIGART International Symposium on Methodologies for Intelligent Systems*, pages 262–269, Knoxville, Tennessee, 1986. ACM.

[47] J. Elgot-Drapkin. Step-logic and the three-wise-men problem. In *Proceedings of the 9th National Conference on Artificial Intelligence*, pages 412–417, 1991.

[48] J. Elgot-Drapkin and D. Perlis. Reasoning situated in time I: Basic concepts. *Journal of Experimental and Theoretical Artificial Intelligence*, 2(1):75–98, 1990.

[49] R. Elio and F. Pelletier. Human benchmarks on ai's benchmark problems. In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*, pages 406–11, 1993.

[50] R. Elio and J. Pelletier. On reasoning with default rules and exceptions. In *Proceedings of the 18th Annual Conference of the COgnitive Science Society*, 1996.

[51] D. Etherington, S. Kraus, and D. Perlis. Nonmonotonicity and the scope of reasoning. *Artificial Intelligence*, 52:221–261, 1991.

[52] D. Etherington, R. Mercer, and R. Reiter. On the adequacy of predicate circumscription for closed-world reasoning. *Computational Intelligence*, 1:11–15, 1985.

[53] David W. Etherington. Formalizing nonmonotonic reasoning systems. *Artificial Intelligence*, 31(1):41–85, 1987.

[54] S. Fahlman, D. Touretzky, and W. Van Roggen. Cancellation in a parallel semantic network. In *Proc. 7th IJCAI*, pages 257–263, 1981.

[55] George M. Ferguson, James F. Allen, Brad W. Miller, and Eric K. Ringger. The design and implementation of the TRAINS-96 system: A prototype mixed-initiative planning assistant. TRAINS Technical Note 96-5, University of Rochester, 1996.

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

[57] Michael Fisher and Richard Owens. From the past to the future: Executing temporal logic programs. In *Logic Programming and Automated Reasoning*, pages 369–380, 1992.

[58] M. Ford and D. Billington. Strategies in human nonmonotonic reasoning. *Coomputational Intelligence*, 16(3):446–468, 2000.

[59] D. Gabbay. The declarative past and the imperative future: executable temporal logic for interactive systems. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Proceedings of Temporal logic in specification*, volume 398 of *Lecture notes in computer science.* Springer-Verlag, 1989.

[60] K. Galotti. Approaches to studying formal and everyday reasoning. *Psychological Bulletin*, 105:331–351, 1989.

[61] A. Garvey, C. Cornelius, and B. Hayes-Roth. Computational costs versus benefits of control reasoning. In *Proc. of AAAI-87*, pages 110–115, Seattle, WA, 1987.

[62] E. Gat. ESL: A language for supporting robust plan execution in embedded autonomous agents. In *Procs. of the AAAI Fall Symposium on Plan Execution*. AAAI Press, 1996.

[63] G. Gazdar. *Pragmatics, Implicature, Presupposition and Logical Form*. Academic Press, New York, 1979.

[64] M. P. Georgeff and F. F. Ingrand. Decision-making in embedded reasoning systems. In *Proceedings of the 6th International Joint Conference on Artificial Intelligence*, pages 972–978, 1989.

[65] A. K. Ghose and R. G. Goebel. Anytime default inference. In *Proceedings of the Fourth Pacific Rim International Conference on Artificial Intelligence*, 1996.

[66] M. L. Ginsberg and D. F. Geddis. Is there any need for domain-dependent control information? In *Proc. of AAAI-91*, pages 452–457, Anaheim, CA, 1991.

[67] M. L. Ginsberg and D. E. Smith. Reasoning about action II: The qualification problem. In F. M. Brown, editor, *Proceedings of the 1987 Workshop on The Frame Problem*, pages 259–287, Lawrence, KS, 1987. Morgan Kaufmann.

[68] F. Giunchiglia, L. Spalazzi, and P. Traverso. Planning with failure. In *Proceedings AIPS 94*, 1994.

[69] P. Gärdenfors. *Knowledge in Flux: Modeling the Dynamics of Epistemic States*. MIT Press, Cambridge, MA, 1988.

[70] N. Green and S. Carberry. A hybrid reasoning model for indirect answers. Association for Computational Linguistics, 1994.

[71] H. P. Grice. Meaning. *Philosophical Review*, 66:377–88, 1957.

[72] H. P. Grice. *Logic and Conversation*. William James Lectures. Harvard University, 1967.

[73] H. P. Grice. Further notes on logic and conversation. In P. Cole and J. L. Morgan, editors, *Syntax and semantics 9: pragmatics*, pages 113–128. Academic Press, 1978.

[74] P. Grice. *Studies in the Way of Words*, chapter Presupposition and Conversational Implicature. Harvard, Cambridge, Ma., 1989.

[75] J. Gurney, D. Perlis, and K. Purang. Active logic and Heim's rule for updating discourse context. In *IJCAI 95 Workshop on Context in Natural Language*, 1995.

[76] J. Gurney, D. Perlis, and K. Purang. Interpreting presuppositions using active logic: From contexts to utterances. *Computational Intelligence*, 13(3):391–413, 1997.

[77] B. Hayes-Roth. Intelligent control. *Artificial Intelligence*, 59(1-2):213–220, February 1993.

[78] I. Heim. On the projection problem for presuppositions. In S. Davis, editor, *Pragmatics*. Oxford, 1983.

[79] I. Heim. Presupposition projection and the semantics of attitude verbs. *Journal of Semantics*, 9:183–221, 1992.

[80] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. C. Meyer. Formal semantics for an abstract agent programming language. In M. P. Singh, A. Rao, and M. J. Wooldridge, editors, *Intelligent agents IV*, pages 215–229. Springer Verlag, 1998.

[81] K.V. Hindriks, F.S. de Boer, W. van der Hoek, and J.-J.Ch. Meyer. Failure, monitoring and recovery in the agent language 3apl. In *Proceedings of the AAAI 1998 Fall Symposium on Cognitive Robotics*, pages 68–75, 1998.

[82] Graeme Hirst and Susan McRoy. The repair of speech act misunderstandings by abductive inference. *Computational Linguistics*, 17(4), 1995.

[83] E. Horvitz and A. Klein. Reasoning, metareasoning and mathematical truth: studies of theorem proving under limited resources. In *Proc UAI 95*, 1995.

[84] E. J. Horvitz. Reasoning about belief and actions under computational resource constraints. In *Proc third AAAI workshop on uncertainty in artificial intelligence*, 1987.

[85] Adele E. Howe. Improving the reliability of artificial intelligence planning systems by analyzing their failure recovery. *Knowledge and Data Engineering*, 7(1):14–25, 1995.

[86] H. Ismail and S. Shapiro. Two problems with reasoning and acting in time. In A. G. Cohn, F. Giunchiglia, and B. Selman, editors, *Proceedings of KR2000*, 2000.

[87] H. Kamp and U. Reyle. *From Discourse to Logic*. Kluwer, Dordrecht, 1993.

[88] L. Kartunnen. Presuppositions of compound sentences. *Linguistic Inquiry*, 4:167–193, 1973.

[89] Paul Kay. The inheritance of presuppositions. *Linguistics and Philosophy*, pages 333–379, 1992.

[90] M. Kifer and E. J. Lozinskii. A logic for reasoning with inconsistency. *J. of Automated Reasoning*, 9(2), 1992.

[91] Mark Klein and Chrysanthos Dallarocas. Exception handling in agent systems. In Oren Etzioni, Jörg P. Müller, and Jeffrey M. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 62–68, Seattle, WA, USA, 1999. ACM Press.

[92] Robert A. Kowalski and Marek J. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.

[93] *The Situation Calculus and Event Calculus Compared*, 1994.

[94] S. Kraus, D. Lehmann, and M. Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial intelligence*, 44:167–207, 1990.

[95] Sanjeev Kumar, Philip R. Cohen, and Hector J. Levesque. The adaptive agent architecture: Achieving fault-tolerance usingpersistent broker teams. Technical Report CSE-99-016-CHCC, 23, 1999.

[96] N. Kushmerik, S. Hanks, and D. Weld. An algorithm for probabilistic least-commitment planning. In *Proceedings of AAAI94*, 1994.

[97] J. Laird and J. Duchi. Creating human-like synthetic characters with multiple skill levels: A case study using the soar quakebot. In *Proceedings of the AAAI 2000 Fall Symposium Series: Simulating Human Agents*, 2000.

[98] J. Laird and M. van Lent. Human-level ai's killer application: Interactive computer games. Invited talk, AAAI 2000.

[99] G. Lakemeyer. Tractable meta-reasoning in propositional logics of belief. In *Proc. of the 10th IJCAI*, pages 402–408, Milan, Italy, 1987.

[100] A. Lascarides and J. Oberlander. Temporal coherence and defeasible knowledge. *Theoretical Linguistics*, 19(19), 1993.

[101] H. Leblanc and W. A. Wisdom. *Deductive Logic*. Prentice Hall, 1993.

[102] H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31:59–84, 1994.

[103] A. Y. Levy, R. E. Fikes, and Y. Sagiv. Speeding up inferences using relevance reasoning: a formalism and algorithms. *Artificial Intelligence*, 97:83–136, 1997.

[104] V. Lifschitz. Benchmark problems for formal nonmonotonic reasoning. In M. Reinfrank, J. de Kleer, M. L. Ginsberg, and E. Sandewall, editors, *Non-monotonic reasoning : 2nd international workshop*, volume 346 of *Lecture notes in computer science.* Springer-Verlag, 1989.

[105] Daniel Marcu and Graeme Hirst. An implemented formalism for computing linguistic presuppositi ons and existential commitments. pages 141–150. International Workshop on Computational Semantics, 1994.

[106] J. McCarthy. Epistemological problems of artificial intelligence. In *Proc. of the 5th IJCAI*, pages 1038–1044, Cambridge, MA, 1977.

[107] J. McCarthy. Formalization of two puzzles involving knowledge. Unpublished note, Stanford University, 1978.

[108] J. McCarthy. Applications of circumscription to formalizing common-sense knowledge. *Artificial Intelligence*, 28(1):89–116, 1986.

[109] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, pages 463–502. Edinburgh University Press, 1969.

[110] John McCarthy. Circumscription–a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[111] John McCarthy. Ascribing mental qualities to machines. In Vladimir Lifschitz, editor, *Formalizing Common Sense: Papers by John McCarthy*, pages 93–118. Ablex Publishing Corporation, Norwood, New Jersey, 1990.

[112] D. McDermott. Planning and acting. *Cognitive Science*, 2:71–109, 1978.

[113] D. McDermott. Non-monotonic logic II. *Journal of the ACM*, 29:33–57, 1982.

[114] D. McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155, 1982.

[115] D. McDermott and J. Doyle. An introduction to non-monotonic logic. In *Proc. of the 6th IJCAI*, pages 562–567, Tokio, Japan, 1979.

[116] D. McDermott and J. Doyle. Non-monotonic logic I. *Artificial Intelligence*, 13(1,2):41–72, 1980.

[117] D. McDermott and J. Doyle. Non-monotonic logic I. *Artificial Intelligence*, 13:41–72, 1980.

[118] Susan McRoy. *Abductive Interpretation and Reinterpretation of Natural Language Utterances.* PhD thesis, University of Toronto, 1993.

[119] Susan McRoy and Graeme Hirst. An abductive account of repair in conversation. In *Working Notes AAAI Fall Symposium on Discourse Structure in Natural Language Understanding and Generation*, November 1991.

[120] Susan W. McRoy and Graeme Hirst. The repair of speech act misunderstandings by abductive inference. *Computational Linguistics*, 21(4):5–478, 1995.

[121] R. E¿ Mercer. Solving some persistent presupposition problems. *COLING*, pages 420–425, 1988.

[122] F. Michaud, G. Lachiver, and C. T. Le Dinh. A new control architecture combining reactivity, deliberation and motivation for situated autonomous agent. In *Proc. of the Int. Conf. on Simulation and Adaptive Behavior (SAB)*, pages 245–254, Cap Cod, 1996.

[123] M. Miller. Reasoning about appearance and reality. Manuscript, 1990.

[124] M. Miller. *A view of one's own past and other aspects of reasoned change in belief*. PhD thesis, University of Maryland, 1993.

[125] M. Miller. *A View of One's Past and Other Aspects of Reasoned Change in Belief.* PhD thesis, Department of Computer Science, University of Maryland, College Park, Maryland, 1993.

[126] M. Miller, D. Perlis, and K. Purang. Defaults denied. Technical Report CS-TR-3680, University of Maryland, 1996.

[127] A. A. Milne. *Winnie the Pooh*. Dutton, 1954.

[128] S. Minton. Is there any need for domain-dependent control information: a reply. In *Proceedings AAAI96*, 1996.

[129] R. C. Moore. Semantical considerations on nonmonotonic logic. In *Proc. of the 8th IJCAI*, pages 272–279, Karlsruhe, Germany, 1983.

[130] Robert Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25:75–94, 1985.

[131] T. O. Nelson. *Metacognition: Core readings*. Allyn & Bacon, 1992.

[132] T.O. Nelson and L. Narens. *Metacognition: Knowing about Knowing*, chapter Why investigate metacognition? Bradford books, Cambridge, 1994.

[133] M. Nirkhe, S. Kraus, M. Miller, and D. Perlis. How to (plan to) meet a deadline between *now* and *then*. *Journal of logic computation*, 7(1):109–156, 1997.

[134] N. Onder, M. Pollack, and J. Horty. A unifying algorithm for conditinoal probabilistic planning. In *Proceedings AIPS98 workshop on Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments*, 1998.

[135] Nilufer Onder and Martha E. Pollack. Contingency selection in plan generation. In *Plan Execution: Problems and Issues: Papers from the 1996 AAAI Fall Symposium*, pages 102–108. AAAI Press, Menlo Park, California, 1996.

[136] J. Pearl. *Probabilistic reasoning in intelligent systems.* Morgan Kaufmann, 1988.

[137] Francis Jeffrey Pelletier and Renée Elio. What should default reasoning be, by default? *Computational Intelligence*, 13(2):165–187, 1997.

[138] J. Pelletier and R. Elio. Logic and computation: human performance in default reasoning. In P. Gardenfors, K. Kijania-Placet, and J. Wolenski, editors, *Logic. Methodology and Philosophy of Science.* Kluwer Academic Press, 2001.

[139] M. Peot and D. Smith. Conditional nonlinear planning. In *Proceedings of AIPS92*, 1992.

[140] D. N. Perkins. *Knowledge as design*. Erlbaum, 1986.

[141] D. Perlis, J. Gurney, and K. Purang. Active logic applied to cancellation of Gricean implicature. In *Working notes, AAAI 96 Spring Symposium on Computational Implicature*. AAAI, 1996.

[142] D. Perlis and K. Purang. Conversational adequacy: Mistakes are the essense. In *Working notes AAAI 96 Workshop on Detecting, Repairing, and Preventing Human-Machine Miscommunication*. AAAI, 1996.

[143] D. Perlis, K. Purang, and C. Andersen. Conversational adequacy: Mistakes are the essense. *International Journal of Human Computer Studies*, pages 553–575, 1998.

[144] D. Perlis, K. Purang, D. Purushothaman, C. Andersen, and D. Traum. Modeling time and meta-reasoning in dialogue via active logic. In *Working notes of AAAI Fall Symposium on Psychological Models o f Communication*, 1999.

[145] Donald Perlis. On the consistency of commonsense reasoning. *Computational Intelligence*, 2:180–190, 1986.

[146] Philips semiconductors. *DSP for CD and DVD ROM systems*, April 2001. SAA7335.

[147] D. Phillips. 2 planes nearly collide at reagan national. Washington Post, June 25 2001. Page A04.

[148] J. Pollock. Rational cognition in oscar. In *Proceedings of ATAL99*, 1999.

[149] D. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36:27–47, 1988.

[150] K. Purang. Alma: an interactive Active Logic MAchine. In *Proceedings of the IJCAI'01 Workshop on Effective AI Resources*, 2001.

[151] K. Purang, D. Purushothaman, D. Traum, C. Andersen, D. Traum, and D. Perlis. Practical reasoning and plan execution with active logic. In *Proceedings of the IJCAI'99 Workshop on Practical Reasoning an d Rationality*, 1999.

[152] W. Quine. *From a Logical Point of View*. Harvard University Press, Cambridge, MA, 1953.

[153] W. V. O. Quine and J. L. Ullian. *The web of belief*. Mc Graw-Hill, 1978.

[154] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[155] Anand S. Rao and Michael P. Georgeff. Modeling rational agents within a BDI-architecture. In James Allen, Richard Fikes, and Erik Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 473–484. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1991.

[156] R. Reiter. On Closed World Data Bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 55–76. Plenum Press, New York, 1978.

[157] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1,2):81–132, 1980.

[158] R. Reiter. A theory of diagnosis from first principles. *Artificial intelligence*, 32:57–95, 1987.

[159] R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy, Vladimir Lifschitz (Ed.), Academic Press.* 1991.

[160] R. Reiter. Sequential temporal golog. In *Proceedings of KR98*, 1998.

[161] R. Reiter and G. Criscuolo. On interacting defaults. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 270–276. AAAI, 1981.

[162] R. Reiter and J. de Kleer. Foundations of assumption based truth maintenance systems: preliminary report. In *Proc AAAI 87*, 1987.

[163] N. Rescher and A. Urquhart. *Temporal Logic*. Springer-Verlag, New York, 1971.

[164] S. Rosenschein and L. P. Kaelbling. A situated view of representation and control. *Artificial intelligence*, 73(1–2):149–173, 1995.

[165] S. Russell and P. Norvig. *Artificial Intelligence: a modern approach*. Prentice Hall, 1995.

[166] S. Russell and E. Wefald. Principles of metareasoning. *Artificial Intelligence*, 49:361–395, 1991.

[167] S. J. Russell and E. Wefald. *Do the right thing*. MIT Press, 1991.

[168] E. Sandewall. *Features and Fluents: the representation of knowledge about dynamical systems*, volume 1. Clarendon Press, 1994.

[169] S. Savignon. *Communicative Competence: Theory and Classroom Practice*. Addison-Wesley, 1983.

[170] K. Schlechta. Defaults as generalized quantifiers. *Journal of Logic and Computation*, 5(4):473–494, 1995.

[171] S. C. Shapiro. An introduction to sneps 3. In B. Ganter and G. W. Mineau, editors, *Conceptual Structures: Logical, Linguistic, and Computational Issues*, number 1867 in Lecture Notes in Artificial Intelligence, pages 510–524. Springer-Verlag, 2000.

[172] Stuart C. Shapiro and William J. Rapaport. The SNePS family. In Fritz Lehmann, editor, *Semantic Networks in Artificial Intelligence*, pages 243–275. Pergamon Press, Oxford, 1992.

[173] Yoav Shoham. A semantical approach to nonmonotonic logics. In *Proceedings of IJCAI-87, 10th International Joint Conference on Artificial Intelligence*, pages 388–392, Milano, IT, 1987.

[174] SICS. *Quintus Prolog Manual*, 1999. http://www.sics.se/isl/quintus/qp/frame.html.

[175] D. E. Smith. *Controlling inference.* PhD thesis, Stanford, 1985.

[176] R. Smith, J. Garstecki, and S. Heijster. Quake III rally sdk. http://www.quakerally.com/docs/, 2001.

[177] S. Soames. How presuppositions are inherited: A solution to the projection problem. *Linguistics Inquiry*, 13:483–545, 1982.

[178] Dan Sperber and Deirdre Wilson. *Relevance: communication and cognition*. Blackwell, Cambridge, Massachusetts, 1995.

[179] T. Strat and J. Lawrence. Explaining evidential analyses. *Intl. Journal of Approximate Reasoning*, 1989.

[180] D. Subramanian and M. R. Genesereth. The relevance of irrelevance. In *Proc. of the 10th IJCAI*, pages 416–422, Milan, Italy, 1987.

[181] D. Subramanian, R. Greiner, and J. Pearl. The relevance of relevance. *Artificial Intelligence*, 97:1–5, 1997.

[182] Elaine Tarone. Some thoughts on the notion of communication strategy. *TESOL Quarterly*, 15(3):285–295, 1981.

[183] D. S. Touretzsky. Implicit ordering of defaults in inheritance systems. In *Proceedings of AAAI 84*, pages 322–325, 1984.

[184] D. Traum and C. Andersen. Representations of dialogue state for domain and task independent meta-dialogue. In *Proceedings of the IJCAI99 workshop: Knowledge And Reasoning in Practical Dialogue Systems*, pages 113–120, 1999.

[185] David R. Traum and James F. Allen. Towards a formal theory of repair in plan execution andplan recognition. Unpublished manuscript.

[186] P. Traverso, A. Cimatti, L. Spalazzi, E. Giunchiclia, and A. Armando. Mrg: Building planners for real world complex applications. *Applied Artificial Intelligence*, 8(3), 1994.

[187] E. van Releghem. Separating control knowledge from domain knowledge. In *Proc. of the 6th ECAI*, page 354, Pisa, Italy, 1984.

[188] P. C. Wason. Reasoning about a rule. *Quarterly Journal of Experimental Psychology*, 20:273–281, 1968.

[189] Webster's third new international dictionary, unabridged, 1993.

[190] Richard W. Weyhrauch, Marco Cadoli, and Carolyn L. Talcott. Using abstract resources to control reasoning. *Journal of Logic, Language and Information*, 1(7):77–101, 1998.

[191] D. E. Wilkins, K. L. Myers, J. D. Lowrance, and L. P. Wesley. Planning and reacting in uncertain and dynamic environments. *JETAI*, pages 197–227, 1995.

[192] David E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann Publishers Inc., San Mateo, CA, 1988.