PARTIAL FORMALIZATIONS AND THE LEMMINGS GAME

John McCarthy, Stanford University, jmc@cs.stanford.edu [*]

1998 Mar 2

Abstract

The computer game *Lemmings* can serve as a new *Drosophila* for AI research connecting logical formalizations with information that is incompletely formalizable in practice. In this article we discuss the features of the *Lemmings* world that make it a challenge to both experimental and theoretical AI and present some steps toward formalizing the game using situation calculus.

Preliminary versions of this paper lack important formulas and references to the literature on reactive AI and to computer vision. The formulas present are not yet integrated into a coherent whole. [1]

[*]Work partly supported by ARPA (ONR) grant N00014-94-1-0775 and partly done while the author was Meyerhoff Visiting Professor at the Weizmann Institute of Science, Rehovot, Israel

[1]This document is reachable from the WWW page http://www-formal.stanford.edu/jmc/home.html.

Contents

1 Introduction

Scientific progress in any field benefits from "models" [2] that allow its phenomena to be studied with a minimum of tedious technicalities and which provide a maximum of scientific information for the work expended. Genetics has used the fruit fly *Drosophila* since about 1910. The motivation for this is not to breed better fruit flies but is associated with technical facts like it being possible to store a thousand fruit flies in a bottle and the generation time being a few days. AI can benefit from having its own simple "models". Here's what looks like a good one that will be useful for solving many important problems in theoretical and experimental AI.[3]

The computer game *Lemmings* (by Psygnosis Ltd.)[Psygnosis 93] provides a framework for formalization of facts about concurrent processes at the level of human interaction with the processes, i.e. when the process with which the human interacts is only partly known and runs too fast for the human to follow. Our examples are all taken from the sampler *Lemmings Jr.*

The paper concerns using *Lemmings* as a model for the logical approach to AI including both theoretical and experimental work, but other approaches may also find it challenging and useful. The designers of *Lemmings* invented games that challenge human ingenuity but keep within a limited *lemming physics* framework that models an interesting part of the *common sense informatic situation.* We argue that AI needs just that kind of challenge.

The problems presented by *Lemmings* are simpler than real-world problems in many important respects. For example, Lemmings is deterministic. The same actions in locally similar situations always have the same local effects. Variations are introduced by limited human ability to repeat the times of actions precisely. However, this is often unimportant.

While much can be learned from theoretical work on formalizing the facts about the *Lemmings* world, getting the full benefit will probably require actual programs to play the game. In principle, experimenting with programs in a domain where human understanding is adequate should not be required. However, there is so much room for wishful thinking about what facts have to be taken into account that the formalizations are unlikely to be adequate unless they guide an actual program.

2 Description of the *Lemmings* Games

A *Lemmings* game is presented as a picture on the computer screen. The picture contains two kinds of objects besides empty space.

[2]Biologists speak of a fruitfly or mouse model of a biological phenomenon. Here we can speak of a *lemmings* model of scene parsing, etc.

[3]Alas, there is in 1994 good reason to write about the importance of basic research models "like a missionary talking to cannibals rather than with a decent terseness and restraint", to use the words of G. H. Hardy [Hardy, 1937].

- solid-objects These include surfaces on which lemmings can walk, cliffs that can be climbed or from which lemmings can fall, hazards like flames or water that kill lemmings that enter them, bridges built by lemmings. Objects other than lemmings do not move, although they may be damaged by explosions.
- lemmings There are ten kinds of lemmings. Each is represented by a characteristic picture of a man about 10 pixels high.

There is a lemming source from which lemmings are emitted until the quota for the game is reached. There is a goal or lemming sink. The object of the game is to get a specified fraction of the lemmings to the goal. If the player doesn't succeed, he gets to try the game over.

All lemmings are initially emitted as *walkers*. Subject to certain restrictions, the player may designate any lemming as having one of the 9 qualities other than *walker*.

Here are the capabilities of the different kinds of lemming.

- walker *Walkers* walk either to the left or to the right at a steady pace on the horizontal or on moderate slopes. They cannot stand still. When a walker comes to a wall it reverses. When it comes to a cliff it falls off. If the fall is from too great a height it dies, i.e. disappears from the scene and therefore doesn't add to the score.
- climber *Climbers* behave like walkers, but when one reaches a wall it climbs it.
- floater *Floaters* behave like *walkers* but can safely fall from any height.
- bomber When a lemming is designated a *bomber*, there is a 5 second countdown, and then it explodes, removing itself and all material from a small circular neighborhood of its location. Explosions do not affect other lemmings.
- blocker When designated as a *blocker*, a lemming stands still until its status is changed by exploding it or otherwise changing its status. Walking lemmings are reflected from *blockers*. Its status cannot be changed by redesignating it.
- builder When designated a *builder* a lemming starts building a bridge diagonally upward in the direction it was walking. The slope is 1/2. Building continues for 12 steps or until the bridge hits something. Lemmings walking to a bridge in the same direction the bridge is being built climb the bridge. A lemming coming to the end will fall off.
- basher When a lemming comes to a wall and is designated a *basher*, it digs a horizontal hole.
- miner A *miner* is like a *basher* except that the hole is diagonally downward.
- digger A *digger* is like a *basher* except that the hole is vertically down.

The above description does not fully describe the behavior of lemming, e.g. how deep a hole has to get before it traps walkers. Properties like this are consequences of the lemming program, are not described in the manual and are imprecisely learned by experience.

The game file bought from Psygnosis contains a number of opportunities to play (10 in the sampler and 120 in the original full version). We call each of them a *game*. The games are organized into levels of increasing difficulty, but we won't use the word level that way; instead we use it in its geometrical sense. Each sequence of actions by the user and the program is called a *play* of the game.

3 What Needs to be Formalized?

Here are some of the phenomena of playing *Lemmings* that are shared by real time, real world problems and which are still not well treated in AI systems.

1. A person playing *Lemmings* is dealing with a process that runs too fast to be fully observed. No-one can follow the motion of 5 lemmings, let alone the 80 common in some levels. Actually, two lemmings is often too much if their images overlap.
2. The laws of motion of the lemming world are quantitative and are characterized by the computer program. However, the player's knowledge of them is qualitative and partial.
3. When the player looks at a lemming scene, he initially extracts into sentences only part of the information he may eventually use to make his decisions. Other information remains in the scene to be used if wanted. This illustrates the partial truth of Herbert Simon's slogan "The world is its own best model". The world is its own most comprehensive model but often not its own most intelligible model.
4. The player's knowledge of a particular situation is partial. Important parts of the scene are often out of sight.
5. There are usually too many lemmings in the scene for them all to be individuated in the player's mind. He has to refer to groups of lemmings—even groups whose composition changes.
6. Each *Lemmings* game has a its own geometry, and the scene must be appropriately parsed into regions of tactical significance, e.g. regions whose points are all accessible by walkers. Digging holes and placing blockers both change the effective division into regions.
7. Nevertheless, successful play also requires that important features of the scene be verbalized and that plans be made. Projection of the effects of carrying out plans is done in a verbal (for computers, logical) language.
8. It is possible to learn from experience and try again. Here *Lemmings* is easier than the real world, because the exact initial situation is repeated when ever a play is restarted. Moreover, *Lemmings* is deterministic. The same sequence of actions will always have the same effects.

People formulate in natural language facts about the *Lemmings* world. We can expect to devise formal tools of equal power. Besides facts, people tell each other lemming stratagems, both local and global. We will not try to formalize the physical skill of making several moves accurately and in quick succession. In the actual game, vigourous use of the pause facility can avoid much of the need for practicing fast reactions.

4 An Example of Lemming Play

At one stage in Game7 in *Lemmings Jr.* the way I first solved it, the situation can be described as follows. See Figure [**?**].

All but one of the living lemmings are in two groups. Some of the lemmings in each group may be still pouring out of a lemming source, but once out the lemmings in each group are bounded by a pair of blockers. These lemmings will go back and forth between the blockers until the player does something about it. In this description nothing is said about the individual lemmings in the two groups. However, when we later give a special role to a lemming in each group, the aggregation changes.

The remaining lemming has parachuted to the lower level and is building the bridges needed for all the lemmings to reach the goal. When this lemming has built the bridges, it will vanish into the goal. Then the player starts a random lemming in each group digging. Eventually each group will fall through to the lower level and march back and forth between barriers. One of these lemmings will be set to building a bridge over the leftmost barrier. When this is done all the remaining lemmings will pass over the barrier and over the bridges built earlier and on to the goal.

The beginning of this game requires more consideration of parallel processes. The first lemming out of the left source is made into a blocker to prevent other lemmings from falling into some water. The second lemming is made into a parachutist and will jump down to build bridges. The third lemming is made into a blocker immediately after it passes the left lemming source after being reflected by the blocker. While all this is being completed, lemmings are coming out of the rightmost source and marching to the right. As soon as the player finishes the tasks on the left he makes the lead lemming on the right into a blocker so as to keep the lemmings marching to the right from reaching something that will kill them. After the rightmost blocker has been installed a left hand blocker for the right hand group is put in place to the left of the right hand lemming source. There is now plenty of time for bridge building.

In the meantime the parachutist is safely marching back and forth between two barriers. He needs to build a bridge over the left barrier, but he cannot reach high enough on this barrier if he starts toward it. Instead he must build towards the right barrier starting from a point that will end a bridge at the right height. Too high and he will go over the right barrier, which isn't wanted. At the right height, when he reverses he can be made to build a bridge toward the left barrier starting at the top of the bridge he has just built. This will get him over the left barrier. After that he must build a bridge over a lemming trap and another bridge up to the slope leading to the goal.

Finally, a lemming in each group is made into a digger. As the hole it digs gets deeper all the lemmings fall into it. Finally it breaks through the ceiling of the lower level, and all the lemmings fall down and can march to goal over the bridges that have been built.

One more tactical fact. All the remaining lemmings from the two sources must be allowed to fall into the holes being dug before the diggers are allowed to break through the ceiling of the lower region. Otherwise, some lemmings will fall through the hole and down to the lower level in one fall, and this will kill them.

The four lemmings used as blockers cannot be saved, and it is convenient to blow them up when the rest of the work has been done.

The above description is accurate for the way I did it then and indicates the kinds of formalizations that my old method required. There are better ways of solving that game, and they require formalizations of different facts about the lemming world and different facts about the specific situations.

5 Physics of the Common Sense Lemming World

The real physics of the lemming world is embodied in the Psygnosis program for playing the machine side of lemming games. Presumably there is a general lemming mover, and the designers of particular games build structures with pictures glued onto them. Let's call it the lemming interpreter, since it also interprets the actions of players. These actions consist almost entirely of designating particular lemmings as having powers selected from the list of 9 possibilities.

However, the human player doesn't know the lemming interpreter, and it wouldn't do him much good if he did. What the human knows is the common sense physics of the lemming world. Our goal is to find an *epistemologically adequate* [McCarthy and Hayes, 1969] way of expressing this physics. Epistemological adequacy requires that the facts that are actually observable be expressible and that the general facts giving the effects of actions be expressed in a way that makes the observations useful.

Here are some aspects of the common sense physics of the lemming world.

1. Lemmings move to the right or left and sometimes climb. I suppose lemmings cannot stand still except for blockers and have only one possible speed. Players use this fact in order to time explosions. If a lemming is triggered while at place x to explode 5 seconds later at place y, triggering at x will always result in an explosion at y.

2. Except for blockers, lemmings don't interact with each other, e.g. they interpenetrate rather than collide.

3. The lemming world has kinematics but apparently not dynamics. Thus the program uses velocities but not forces and accelerations. Our own projections of

plans is rarely even kinematic but merely geometric. By non-kinematic I mean that the velocities of the lemmings are not used in projection.

4. Lemmings fall at constant velocity but are destroyed if they fall from too great a height.
5. Bridges are infinitely strong and don't break under the weight of any number of lemmings.
6. Explosions kill only the lemming that has been told to explode. Others are unharmed even if they are within the explosion zone. Structures that seem to have been supported by structure that is destroyed do not collapse but "hang in the air".
7. There seems to be no random element in lemming games except that randomness coming from the human player. As is usual in sports, this permits the human to learn precise procedures from repeated plays of a game. Thus he can learn the best place to start a bridge.
8. *Lemming time* is sometimes important. For example, a digger needs a certain amount of lemming time before the hole is deep enough to trap other lemmings. Before that, a walker falling into the hole will climb on out of it. On a larger scale, the total lemming time available for a game sometimes affects strategy.

6 The Problem of Formalization

The *Lemmings* player cannot succeed only by reacting directly to the current situation. He [4] must plan ahead, often to the end of the game in which the situation will be very different from the situation when the plan is made. We propose to do this by formalizing the relevant facts in logic and projecting by logical reasoning. I can't prove there is no other way of doing the projection.

Here are some requirements for projection.

1. The ontology (set of entities over which variables range) of the *Lemmings* world includes lemmings, sets of lemmings, regions, lemming traps, bridges, events including actions, situations, fluents and others. I suppose we should include in the ontology the predicate and function symbols we shall need in the formalism.
2. The events that create, destroy and transform the different kinds of object need special treatment.
3. There must be names for sets of lemmings determined by some properties without the user having to know particular lemmings or even how many there are in the set. Naturally, sets with one element play a special role.

[4]The pronoun "he" embraces women, men and robots.

Maybe the only kind of lemming that has to be treated in sets is *walker*. Others can be treated individually, because only a few can exist at a time. What is likely to be interesting is the set of walkers in a region. The events that happen to sets of walker as sets need to be distinguished, e.g. a set of walkers can fall into a hole. If each walker in a set falls into a hole, then the set falls into the hole. (If you like old jargon, falling into a hole is an intensive property like temperature.) However, we should be able to draw conclusions about what happens to a set of lemmings without always having to reason about its members. The cardinality of a set is important when a certain resource, e.g. the number of lemmings that can be designated as climbers, is limiting.

4. Natural language discussion of *Lemmings* suggests that we provide for fluents denoting continuous action, e.g. we should be able to say $holds(walking(l, c0, R), s)$, meaning that lemming l is walking to the right in region $c0$ in situation s. We will also want to say $holds(walking2(l, c0), s)$, omitting the direction. The former is stopped by encountering a wall, and the latter is not stopped by encountering a wall but is stopped by l becoming a digger or falling into a hole dug by a digger.

5. There must be a way of dealing with parallel processes. “While I am making group A of lemmings do this, group B is doing that.”

6. Counterfactuals are used in human reasoning about *Lemmings*, e.g. “if I had one more bridge-builder I could ...”.

The formalism must be *elaboration tolerant*. A person plays *Lemmings* with an incomplete knowledge of the properties of the kinds of lemmings and their interaction with features of the scene. An adequate formalization has the property that it can be elaborated to take into account new phenomena by adding statements rather than by completely redoing it.

7 Objects in Space in the Lemming World

Space in the lemming world is two dimensional. Nevertheless, an epistemologically adequate representation of space and objects for the *Lemmings* world requires features that go beyond what has been treated in the AI literature.

The screen in the *Lemmings* world is a pixel map. Like all modern computer games it uses colors, but the literature says *Lemmings* is playable on a gray scale display. We suppose this pixel map is accessible to a program that plays the game. To play the game it also needs the ability to imitate moving and depressing the mouse. There are keyboard shortcuts for some of the mouse actions, but we ignore them, because we are not trying to formalize human speed of action.

The problem for AI is that a human player does not decide what to do operating directly from the pixel map. Instead the human considers the screen as divided into *regions*, some of which are *chambers* consisting of empty space that can contain lemmings and possibly other objects and some of which are solid material. Some of the solid material can be turned into space by digging. Separated regions are sometimes combined when lemmings build bridges.

Human players do not convert the picture into an internal form from which all decisions are made without further reference to the picture. Information from the picture is put into mental forms that are not well understood physiologically or psychologically, and decisions about what to do involve repeated reference to the picture. In particular, one often looks at the details of particular areas of the scene or puts newly relevant global features into the mental model. On the other hand, important facts about the picture can be internalized well enough so that I player can get a new idea about how to win while out of sight of the game.

We claim that these aspects of the human way of handling visual information are not just peculiarities of humans. Their main aspects are features of what we have called the *common sense informatic situation* [McCarthy, 1989]. Computer programs operating in the real world, e.g. controlling mobile robots, also face the common sense informatic situation. This is characterized by incomplete information about the situation itself and incomplete information about the laws that determine the effects of actions.

Therefore, we plan that a *Lemmings* program will parse the scene into a collection of regions and relations among them. The initial parsing will be later extendable by further reference to the pixel map. Like the human the program will use the scene as its own most comprehensive model, although not always its most intelligible or useful model.

The parsing therefore has two unusual aspects.

1. The parsing is not intended to be conclusive. The parser may be asked to get more detail about something it returned previously.
2. The parser looks for common simple regions which it may have to elaborate later rather than having a universally applicable scheme for parsing anything.
3. The parser gives names to features it finds and generates sentences characterizing them. These sentences are later used by reasoning programs that decide what to do.

   The simplest kind of region may be called a chamber. The simplest kind of chamber has an essentially horizontal floor and walls at each end that reflect walkers.[5]

In section 8, page 12, we will discuss actions and events using situation calculus. $S0$ stands for the situation when a particular game called Game0 starts. Game0 is a

[5]Within regions the horizontal location of a position may be important. For example, it usually matters where a bridge is started within a region. I remember positions by irregularities in color or relief. Should we be purist and require this of programs, or should we let them use co-ordinates?

simplification of the first game in *Lemmings Jr.* There is only one lemming. See figure [**?**].

A simple chamber will be rectangular on the screen with walls at both ends. We make $c0$ a simple chamber in $S0$ and can rely on persistence to keep it a simple chamber in $S2$ and beyond. This would be used in planning to solve Game0. Alternatively, if we are actually playing Game0, we can observe that $c0$ is a simple chamber in $S2$. This requires a mechanism for associating names with features of the picture and with situations depicted on the screen. We haven't decided how to do this yet.

Consider chambers which confine lemmings until something is done about it. They are bounded by simple closed curves, but that high level of generality isn't the common sense way of thinking about them. One could suppose that chambers in the lemming world are a subclass of the interiors of simple closed curves and try to formalize a suitable subclass. This is a bad idea, because we haven't seen all the chambers Psygnosis Ltd. has chosen to use, and they are likely to invent new ones with each new lemming variant they market. Their artists draw the chambers and the regions that surround them according to the requirements of the game designers and their artistic taste. These drawings are then digitized. No human player can represent a digitized screen pixel by pixel, and our formalism can't do this either.

Instead we start with simple figures and elaborate them. The *simple chamber* is one of these starts. A program that reasoned about the game would observe the screen of Game0 and recognize the upper chamber as a simple chamber and give it a name. In Game0, the floor of $c0$ is not entirely even, but these surface rills make no difference. It doesn't matter where $l0$ digs its hole. In other lemming games it does make a difference. For example, it might have been necessary in Game0 to dig in a low place in order that the lemmings should not fall too far.

We propose to handle unevenness by allowing additional statements about chambers that have been identified as simple chambers. Complete descriptions of surfaces are not required, but relevant observations may be expressed. Perhaps this is a form of belief revision, e.g. the player formerly believed the chamber was simple and has changed hsi mind; some of the considerations advanced in the study of belief revision might then apply. Another way of handling this idea is to say that $c0$ is a certain kind of simple chamber and express the unevenness in describing what kind.

We also will handle the chamber with the digger's hole by elaborating the simple chamber description. This will be belief update rather than belief revision.

We can implement the above using a notion of chamber type, or more generally, type of lemming region. *simple-chamber* would be the type that is instantiated by $c0$. $at\ right\text{-}end, lemming\text{-}sink, simple\text{-}chamber)$ is is instantiated by $c1$. (We are not committed to this particular notation for elaborating simple figures.) We need types in the formalism and not just individual regions, because we need to make new types by elaborating simpler ones, e.g. we make a chamber with the lemming sink at one end by elaborating simple chambers.

7.1 Geometric Objects and Relations

Here are some useful geometric objects and relations.

1. Two dimensional objects—regions including chambers. We have already discussed them.
2. One dimensional objects—segments and walls. A segment is a surface over which lemmings can walk. Mostly we are concerned with segments bounded by walls or drop-offs or other impediment to walkers. However, segments bounded by being above a hill at a lower level may need to be an object because digging in that segment has a different result than digging elsewhere.

   On walls useful segments are bounded by overhangs as well as floors and ceilings, because climbers can't climb overhangs.
3. Bridges and segments of bridges require special treatment, because they are dynamic until completed. Even when it is complete, it is sometimes necessary to break a bridge into two segments using a digger.
4. The adjacency relations and containment relations among geometric objects are used.

7.2 Geography of Game1

The basic representation of a *Lemmings* scene is the pixel map provided by the *Lemmings* program. However, additional representations are needed for the following purposes.

1. Prediction of what will happen. When a person does it, the person does not generate an new pixel map. To do that the person would need to be able to simulate the *Lemmings* program, and we can't do that. Apparently prediction is based on a representation of scenes simple enough for a person to update.
2. Planning a strategy.

[the actual geography to be provided]

7.3 Geography of Game7

[to be provided]

8 Predicting the Future

Prediction is difficult—especially of the future.

We use situation calculus.

$holds(p, s)$ means that the proposition (i.e. propositional fluent) p holds in situation s.

$value(exp, s)$ is the value of the expression exp in situation s.

$F(p)$ is the propositional fluent that proposition p will hold in the future. F is a reified temporal logic operator. We usually omit parentheses for functions of one argument so that $F(fluent(p))$ would be written $F\ fluent\ p$.

$next(p, s)$ is the next situation after s in which p holds. It is related to F by the axiom

$$holds(F\ p, s) \supset holds(p, next(p, s)). \tag{1}$$

The function F plays the role of a temporal logic future operator. $next(p, s)$ denotes the next situation after s in which p holds. We intend to say nothing about the value of $next(p, s)$ when $F\ p$ doesn't hold.

[An alternative is to regard $next(p, s)$ as an *individual concept* as discussed in [McCarthy, 1979b]. We would then write

$$exists\ next(p, s) \supset holds(p, next(p, s))$$

as the axiom.]

We also have a conditional future function $F'\ p$. The intent is that it predicts that p will hold if nothing to prevent it happens. There will be an axiom of the form

$$holds(F'\ p, s) \wedge \langle\text{nothing prevents } p\rangle \supset holds(F\ p, s). \tag{2}$$

It isn't yet clear how to write the condition $\langle$nothing prevents $p\rangle$.

Here is one way.

What will surely occur is any p such that $F'(p, s)$ and such that there is no p' occurring earlier and preventing p. Therefore, we have

$$F'(p, s) \wedge (\forall p')\neg(p' \neq p \wedge F'(p', s) \wedge time(p', s) < time(p, s) \wedge prevents(p', p)) \supset F(p, s) \tag{3}$$

We can put this in a form that is less explicit about time, namely

$$F'(p, s) \wedge (\forall p')\neg(p' \neq p \wedge F'(p', s) \wedge precedes(p', p, s) \wedge prevents(p', p) \supset F(p, s) \tag{4}$$

Another possibility is to consider the fluents p as having associated times or at least a temporal order. This makes them *richer* objects than we have previously taken them to be. The equations then become

$$F'(p, s) \wedge (\forall p')\neg(p' \neq p \wedge F'(p', s) \wedge time(p') < time(p) \wedge prevents(p', p) \supset F(p, s) \tag{5}$$

and

$$F'(p,s) \wedge (\forall p')\neg(p' \neq p \wedge F'(p',s) \wedge precedes(p',p) \wedge prevents(p',p) \supset F(p,s). \quad (6)$$

Probably the basic inference task is predicting what happens in a situation when there is no intervention. This is just predicting what would be observed by running the program, but predictions used for planning need to be done in an epistemologically adequate way, i.e. using only the information that would be available to a human player. The result will be qualitative, sometimes uncertain and sometimes incorrect.

The basic fact about lemmings walking can be formulated in first order logic as follows.

$$\begin{array}{ll} & holds(rightist\ l, s) \\ \wedge & holds(at(l, p1), s) \\ \wedge & holds(on(p1, surface1) \\ \wedge & holds(on(p2, surface1) \\ \wedge & holds(to\text{-}the\text{-}right\text{-}of(p1, p2, surface1), s) \\ & \neg(\exists x) holds(prevents\text{-}walk(x, p1, p2), s) \\ & \qquad\qquad \supset \\ & holds(F\ at(l, p2), s), \end{array} \quad (7)$$

where

$$holds(prevents\text{-}walk(x, p1, p2), s) \equiv prevents\text{-}walk1(x, p1, p2, surface1, s) \quad (8)$$

and

$$\begin{array}{ll} & (\exists p')(holds(between(p1, p', p2, surface), s) \\ \wedge & holds(at(x, p'), s) \\ \wedge & holds(stops\text{-}walkers\ x, s) \\ & \qquad\qquad \supset \\ & prevents\text{-}walk1(x, p1, p2, surface, s). \end{array} \quad (9)$$

We have made *prevents-walk* a term so it can be an argument of *holds*, whereas *prevents-walk*1 is a predicate so it can be circumscribed. Maybe there will be some way to get rid of this bureaucratic complication.

We would circumscribe the predicate *prevents-walk*1, and if there were no obstacle, we would be able to infer that lemming l would indeed reach point $p2$.

Additional phenomena that might prevent walking to a destination are to be treated by adding sentences whose conclusion is $prevents\text{-}walk1(x, p1, p2, surface, s)$, i.e. the conclusion of (9).

The above equations are unpleasantly long, and in more complicated cases might get quite a bit longer. In dealing with situations that actually arise in playing a game, these equations would be paralleled by a program that would establish that lemming l would reach $p2$ in the concrete situation. In subsection 8.2 we discuss how a program can be connected to a predicate or function in such a way that when it is necessary to reason logically about the predicate applied to constant arguments the program can be used to get the value. We call this attachment of the predicate to the program. It is a special case of partial evaluation.

We might hope to use program in more abstract cases, e.g. thinking about hypothetical situations, like "if I had a blocker there ...". Constant symbols and construction functions would be needed to construct the a representation of the hypthetical situation itself and the other constants on which the program would operate. Presumably the hypthetical situations would be *objects* in the sense of object-oriented languages, e.g. CLOS.

However, only equations retain *elaboration tolerance*, e.g. permit the introduction of new lemming properties by making an assertion.

8.1 Persistent Behaviors

Tom Costello [?] has proposed generalizing the notion of persistence. Instead of a fluent persisting, he proposes that a *fluxion* (a further borrowing from Newton) persists. A fluxion is a function of time. The following is an adaptation of his idea.

Consider a lemming walking. Consider the following formulas:

$$occurs(start\,walking, s) \supset holds(walking, s) \tag{10}$$

and

$$continues(walking, s1, s2) \supset value(x, s2) = value(x, s1) + v(time(s2) - time(s1)), \tag{11}$$

where v is the velocity of the lemming, and we have the definition

$$continues(p, s1, s2) \equiv (\forall s)(s1 \leq s \leq s2 \supset holds(p, s)). \tag{12}$$

walking is thus treated as an ordinary persistent fluent in (10). Its effect on the position of the lemming is given by the auxiliary formula (11).

A persistent fluent continues to hold until something stops it, and then it ceases to hold.

$$\begin{aligned}&\mathit{persistent}\ p \wedge \mathit{holds}(p,s) \wedge \mathit{holds}(F\ \mathit{stops}\ p, s)\\ &\qquad\supset\\ &\quad\neg \mathit{holds}(p, \mathit{next}(\mathit{stops}\ p, s))\\ &\quad\wedge(\forall s')(s \le s' \le \mathit{next}(\mathit{stops}\ p, s) \supset \mathit{holds}(p, s'))\end{aligned} \tag{13}$$

For completeness, we should also include

$$\mathit{persistent}\ p \wedge \mathit{holds}(p,s) \wedge \neg\mathit{holds}(F\ \mathit{stops}\ p, s) \supset (\forall s')(s \le s' \supset \mathit{holds}(p,s')), \tag{14}$$

but there isn't much that lasts forever.

8.2 Attachment

A computer program doing logical inference need not compute $53 + 18$ by deriving $53 + 18 = 71$ from the laws of arithmetic. Instead it should take advantage of the computer and get this result from the computer's arithmetic. This is accomplished in various interactive theorem provers by *attaching* a program, in Lisp for instance, to the function name, in this case $+$. The following considerations apply.

1. As a proof step, the prover is given an expression to evaluate, e.g. the command is

   $$\mathit{evaluate}\ \text{“}53 + 17\text{”}. \tag{15}$$

   The result of the evaluation is a sentence, e.g. $53 + 17 = 71$, which can then be used for further inference like any other logical sentence.
2. The most straightforward evaluators can only be given ground expressions to evaluate. Fancier ones can do algebraic and logical simplification of expressions involving variables. For *Lemmings* we contemplate only the ground case.
3. For *Lemmings* the most important expressions to evaluate will involve the scene as an argument.
4. The attachments are done by the builder of the system, and therefore have the same inferential status as the axioms. In principle, one might use a program verification system to show that the functions defined by the program satisfy the axioms involving the function names. I am not aware that this has ever been done.
5. As described in [McCarthy, 1962], attachments can be combined with reflexion principles to allow the results of decision procedures to be entered as formulas. I don't think *Lemmings* will require this.

Richard Weyhrauch [**?**] has the most elaborate system involving attachment. I don't think *Lemmings* will require its complexities.

9 Programs for Playing *Lemmings*

How can the above facts and proposed formalisms be used in a program for playing *Lemmings*?

9.1 Planning

The simplest kind of planning for *Lemmings* is to give a sequence of objectives, each of which is to be achieved reactively on the basis of the scene visible at the time of completion of the previous objective. Let's see how this works out for Game7 of *Lemmings Jr.*. We give one such plan but note where there may be some problems with a purely reactive system for achieving the successive objectives.

In Game7 there are two lemming sources that open approximately simultaneously, both above a platform. In the middle of the platform there is a pond that drowns lemmings that fall in. It might be surmounted by a long bridge, but none of the solutions I have attempted involve doing this. As in all *Lemmings* games, the lemmings walk to the right when they drop out of the source. The lemmings must be got down from the platform and then marched to the left over or through various obstacles. Here's the plan.

1. Start first left lemming (lemming from left lemming source) digging just after it has moved away from under source.
2. Start first right lemming digging just after it has moved away.
3. Start any lemming that over-runs the hole being dug to digging its own hole. There won't be more than two.
4. If a left lemming bounces left (happens rarely) make it a floater.
5. Increase rate of lemmings coming from source to maximum.
6. Make left digger a basher before it breaks through ceiling.
7. Make first right digger a miner as soon as its hole is deep enough so that lemmings won't climb out of it.
8. When left basher stops at iron wall, make it a digger.
9. When left digger digs far enough to clear iron wall, make it a miner.
10. **Comment:** These steps will get the lemmings safely down. There are now two cases according to whether a jumper is between the two mounds on the lower level or not.

    If yes, go to step 12
11. Make first lemming to approach the right tower from right into a climber.
12. Make sure the lemming going to the left between the two towers is a climber going left.

13. When the climber has passed the left tower, make it a builder at a place where it will build over the lemming trap in one round of building. The right place is determined by experience. If the most obvious place is chosen, the monster reaches out of the trap. If experience is to be used, then there must be a memory of places from play to play of the game. Otherwise, the place must somehow be designated by the planner to the reactive player. When I am playing the game, I remember landmarks like bushes, but it would be easy to devise a co-ordinate system and give the x-co-ordinate. This might be considered contrary to the idea making the program solve the problem that humans solve, using very limited capabilities of numerical estimation.
14. When the builder has passed the lemming trap and become a walker again, make it a builder again when it has reached the point where a two section bridge will reach the slope leading to the goal with the lemming continuing to the goal. Again experience is involved to learn the right place. Again measurement might replace experience, since a *Lemmings* bridge rises on pixel for every two pixel it goes horizontally.
15. When the builder has reached the end of the first stage of bridge building, make it a builder again. (A two stage bridge is needed and suffices.)
16. When the builder has reached the slope, designate a lemming on the right as a basher, just as it reaches the right tower going left.
17. When the basher has broken through and become a walker, designate it a basher again when it reaches the left tower.
18. This is all that needs to be done. All the lemmings are saved.

We believe that each of the steps mentioned above can be acomplished by reactive programming except that step 12 involves remembering whether it is necessary to designate the lemming between the towers a climber. If it was the jumper, then it is necessary, but if it just climbed it is unnecessary. The conditional element could be avoided in this case, because it does no harm to designate a lemming a climber that is already a climber. However, it is probably a good idea to allow reactive programs some reference to memory.

In this example, it seems that all the reactive execution is done subordinate to single steps generated by the plan. It is not clear whether this is always possible in *Lemmings.* Moreover, it makes the plan rather rigid. There is no provision for going back to the planning level if the reactive level encounters an unexpected difficulty.

Consider the communication between the logical planner and the reactive executor of steps. The following considerations apply.

1. Approximately as proposed in [McCarthy, 1959], when the planner infers a sentence of the form

$$\textit{should-achieve } p \tag{16}$$

as a side-effect, the reactive achiever is called with p as argument. When the goal is achieved, the observation program takes another look at the scene and modifies the logical database accordingly. Inference then resumes.

2. Any information needed to achieve a goal that cannot be obtained by the achiever directly from the scene must be provided to the goal-achiever in advance.
3. This scheme provides for no other communication between the logic part of the system and the reactive part and is therefore less sophisticated than many existing systems that plan and execute. At least conceptually, however, it seems appropriate to begin with this primitive interaction and then study what more this particular fruit fly may require.

9.2 First Experiments with Lemmings

Perhaps I am too unambitious, but *Lemmings* strikes me as presenting considerable experimental difficulties, and I advocate starting slowly. Here are some steps.

1. A tool is needed for programs that interact with the *Lemmings* program. It should begin with interaction with the pixel maps displayed by the program. Programs like Planet X bring the pixel map of a Macintosh into an X-window, and an analogous program could bring it into an X-window reachable by a Common Lisp or C++ program. Alternatively, the *Lemmings* pixel map might be accessed directly by programs in the Macintosh or in a PC.
2. Another early step is the parsing of the pixel map into regions. The result of such a parsing is a tree of regions and subregions. Very likely the parsings will have to be extendable when the program decides that it is necessary to go back to the pixel map for more information.
3. The first programs to play the game should concentrate on the easier games. Many of these can be solved greedily, i.e. by reacting to the scene with actions that move the situation closer to the goal.
4. The next step is tree search as discussed in the next subsection.

9.3 Tree Search

Perhaps *Lemmings* games can be solved by tree search. If so, it will be a very sophisticated tree search.

First consider a brute force tree search. The actions available to a player can be described as clicking the mouse on a point on the computer screen or doing nothing. An action must be taken at an interval which is less than one second but probably not less than 0.1 seconds. Some parts of the screen are unresponsive, so we can regard doing nothing as clicking on such a place. The screen on a Macintosh PowerBook is 640 by 480 pixels. Thus the player has 307200 choices every 0.1 seconds. Even if the player could fully observe the consequences of each choice, the branching would make tree search of the full tree impractical.

What must we do to make tree search practical? We first consider what reduced trees to use.

1. First we merge clicking on a quality and clicking on a lemming. This eliminates clicking on two qualities in succession which is pointless, because it is equivalent to clicking on just the second quality.
2. Now we choose a delay. The most common delay is 0. Most things you want to do are done right away. We want to avoid branching on times, so we choose times by events. Thus a clicking on a lemming x the next time lemming y is in a particular position is a choice. Naturally, the case when x is the same lemming as y is distinguished. We still have too much branching, and we must avoid even generating far-off events, at least until earlier parts of the tree have been searched.
3. Now we choose the lemming that is to receive the quality. For this search to be efficient the lemmings must be grouped into equivalence classes, so that choosing only one lemming in the class is considered. Indeed when the lemmings are close-packed, it isn't possible to tell one walker from another.
4. A common type of event is a lemming, usually the one we intend to modify, reaching a certain place. If the lemmings move a pixel at a time (I'm not sure of that.), then there may be several hundred places where we could choose to click. Many consecutive choices of place are likely to be equivalent. In order to reduce the choices the horizontal or vertical line along which the lemming is moving has to be divided into heuristically different regions.

   Here is where a real difference with previous tree search algorithms must arise. We need to make the division into regions tentative and refine it when this is required.
5. In one game a blocker is needed, but 100 percent of the lemmings are to be saved, and therefore having to blow up the blocker will result in a loss. The tree search, if that's what it is, is abstract, because the node in which there is a leftover blocker is at the end of the game. The solution is that the very first lemming out of the source is made into a floater (parachute jumper) and then into a blocker; the very last lemming out of the source is made into a floater and then into a miner just when it can undermine the blocker. Both then parachute to safety.

   If we regard this reasoning as tree search, then we consider making the first lemming a blocker as one edge, solving the rest of the game as another edge, see this as a loss, backtrack and put in making the lemming into a floater, etc.
6. In building a bridge or digging a hole, the precise place to start is sometimes important.

The tree search algorithm itself has to decide in what order to examine the nodes. This clearly requires many heuristics, most specifically considering first moves and sequences of moves that are recommended by some idea. This is where logic will surely creep back in.

It is not obvious that all the above notions can be applied in a way that will reduce the tree search to managable proportions. My intuition is that they can—at least for large parts of many games.

Tree search often becomes inefficient when the sequence of actions is divisible into actions that independently affect different aspects of the situation. Then it usually becomes more efficient to break the problem into parts that can be thought about separately, solve the separate problems and then combine the results. I think *Lemmings* has enough of this to make a pure tree search algorithm impractical.

(This phenomenon is what makes pure tree search algorithms impractical for Go. It is necessary to consider the different areas of the board separately and then combine the results.)

Lemmings tree searches can include retries of games. For example, when the place a blocker is exploded turns out to be too thin, the program should take another look at the pixel map and find a thicker place to explode the blocker.

10 Some Formulas for a Simple *Lemmings* Game

Game0 is a simplification of the first game of *Lemmings Jr.* There is only one lemming.

We give a sequence of formulas narrating the journey of a lemming $l0$ from the lemming source to the lemming sink. It applies to the journey of the lemming chosen to dig even when other lemmings are present. For each formula we say where we expect it to come from.

$$holds(F\ open\ source, S0) \tag{17}$$

As is conventional in situation calculus formalizations, $S0$ is the initial situation. The formula states that the lemming source will open. This is inferred from a general fact about the start of lemming games. Some lemming games have more than one source. However, we may take it as an observation of the screen that there is one lemming source in this game.

Let $S1 = next(open\,source, S0)$. We now have

$$holds(F\ falls\ l0, S1).$$

This follows from (8). It is a general fact about *Lemmings* that a lemming will fall out if not all have fallen out. Calling one of them $l0$ is allowable, but we'll do this more precisely in a later version of these formulas.

Let

$$S2 = next(falls\ l0, S1). \tag{18}$$

This is the next situation in which $l0$ is falling.

$$holds(simple\text{-}chamber\ c0, S0)$$

A simple chamber will be rectangular on the screen with walls at both ends. We make $c0$ a simple chamber in $S0$ and can rely on persistence to keep it a simple chamber in $S2$ and beyond. This would be used in planning to solve Game0. Alternatively, if we are actually playing Game0, we can observe that $c0$ is a simple chamber in $S2$. This requires a mechanism for associating names with features of the picture and with situations depicted on the screen.

Next we want

Let $p0 = value(below\ source, c0), S0)$.

This is the point on $c0$ immediately below the source. I suppose we should assert (and later infer) it exists before we name it.

$$holds(above(source, floor\ c0), S0).$$

$$(\exists p)(holds(above(source, p), S0) \wedge holds(on(p, floor\ c0), S0).$$

Now we can have

$$p0 = value(below(source, c0), S0).$$

We are now ready for

$$holds(F\ at(l0, p0), S2)$$

and consequently

$$holds(at(l0, p0), S3)$$

with

$$S3 = next(at(l0, p0), S2).$$

Now that $l0$ is down on the floor of $c0$, it will walk around until the player turns it into a digger. The details of that, while part of the "biography" of $l0$, do not come into postulating

$$holds(F\ digger\ l0, S3),$$

because this refers to an action of the player in designating $l0$ to be a digger. From it we get

$$S4 = next(digger\ l0, S3).$$

The fact that $l0$ is still in $c0$ does come into inferring from

$$p1 = value(location(l0), S4)$$

that

$$holds(on(p1, floor\ c0), S4).$$

Now we need more geography, including the lower chamber $c1$. We need to say that $c1$ is a simple chamber modified by putting the lemming sink at its right end. We also must say that every point of the floor of $c0$ is above the ceiling of $c1$.

I am not satisfied with the suggestions given above for describing chamber types as modifications of basic types, so we will omit this for now. On the other hand, it is straightforward to say that the floor of co is above the ceiling of $c0$. We have

$$(\forall sp)(holds(on(p, floor\ c0), s) \supset (\exists p')(holds(on(p', ceiling\ c1), s) \wedge holds(above(p, p'), s)))$$

and from it we get

$$p2 = value(below(p1, ceiling\ c1), S4)$$

and

$$S5 = next(at(l0, p2), S4).$$

We have used $holds(diggable(p1, c1), S0)$.

Similarly we want to infer that $l0$ will fall to a point $p3$ on $floor\ c1$, giving

$$S6 = next(at(l0, p3), S5).$$

$$holds(walker\ l0, S6)$$

Now we have two cases according to whether $holds(rightist\ l0, S6)$ or $holds(leftist\ l0, S6)$. In either case we can show

$$F(at(l0, lemming\text{-}sink), S6)$$

.

I used $holds(p, s)$ instead of just $p(s)$, because we might want to quantify over p. However, we haven't used this in the formulas given so far.

There are many more non-trivial matters that require careful consideration.

10.1 Next Steps

*Game*0 is simple enough so that it is almost a conventional situation calculus problem. Even *Game*1 requires more.

1. *Game*1, the first game of *Lemmings Jr.* has 10 lemmings. We must introduce an object for the group of them, and then separate out the lemming designated to be the digger.
2. When the hole is shallow, lemmings pass over it. When it gets deep the fall in. They continue to alternate directions while in the hole, and the direction a lemming has when it falls out of the hole determines which way it will walk on the floor of $c1$.
3. Even in *Game*0, the sentences about the future are derived by nonmonotonic reasoning. We haven't said how this is done. The simplest aspect of this is that the geometric relations of the chambers tend to persist, but the positions of the lemmings don't.
4. Continued events like digging or falling need both macroscopic and microscopic formalizations. At a macroscopic level, a lemming falls to the floor or digs to the ceiling of the lower chamber or walks to the end of the chamber. At a microscopic level the falling or digging is continuous and can be interrupted or can cause other events part way.
5. The geography involves describing the scene to the extent needed as as collection of related instances of spatial types. The formalization must tolerate elaborations like the change in the floor of a chamber caused by a basher (horizontal digger) plowing it up. When an elaboration is not required, it shouldn't affect the statements that are made or the reasoning done with them.
6. Concurrent action needs to be describable in whatever level of detail is needed to draw the required conclusions.
7. What a person learns and can communicate to another person is often that in a certain kind of situation certain changes *can* be achieved. Therefore, to be able to give advice to a *Lemmings* program will require a formalization of *can.* Something is known about this problem, less about how to formalize *how* something that can be done is to be accomplished.

11 Remarks

Some of the ideas in this section may have to be abandoned. Some of the ideas discussed informally in the present draft may be included in the more formal sections of the paper.

11.1 Nonmonotonic Reasoning

Lemmings may present new problems for nonmonotonic reasoning.

1. It is a general feature of nonmonotonic reasoning to implicitly infer that the facts being taken into account are all that are relevant to the phenomenon being reasoned about. However, this has not been put into the formalism explicitly. Making it explicit may be important for reasoning about *Lemmings*. A *Lemmings* player learns to win a particular game by repeated trying. He uses the fact that the same action in the same situation always has the same result. Formally this is assumption is incorporated in the use of a function *result* to get a new situation $s' = result(e, s)$, i.e. the new situation is determined by the old situation and the event (action) that occurs in the old situation.

   However, a player uses more than the functionality of the result of an event, because he wants to learn not merely from exact repetitions of the situation but from repetitions of those aspects of the situation that are relevant to the phenomenon he is learning about.

   Therefore, we need to be able to express as a circumscription jumping to the conclusion that we have repeated the relevant aspects of a previously examined situation. In *Lemmings*, this is related to locality, i.e. it doesn't matter what the lemmings elsewhere in the scene are doing.

2. Something like the philosophers' paradox of the heap comes up in *Lemmings*, and a solution suggests itself that may have more general application. Suppose a lemming is removed from a bunch of lemmings. We draw the nonmonotonic conclusion that we still have a bunch. If you repeatedly draw this conclusion enough times, one of the conclusions will be false, because you will have eliminated the whole bunch. So what? The reasoning was nonmonotonic and admittedly risky.

   There is a related fact about nonmonotonic reasoning. Suppose for a variable x you infer nonmonotically $p(x)$. Your nonmonotonic logic should be such that this does not allow you to infer $(\forall x)p(x)$, i.e. universal generalization is not allowed.

11.2 Maybe projection isn't all Logic.

In AI projection can be done by running the progam, provided the complete state of the system is available, and the program describing the evolution of the system is available.

Suppose the neither a complete description of the situation nor a complete description of the process is available. Situation calculus ([McCarthy and Hayes, 1969]) was invented to deal with this by allowing logical inference from facts about a situation for which we do not have a complete description.

However, it is possible that humans do some projection by non-logical means, i.e. by running some mental process on a representation of the situation. If so, such mental simulation must use a greatly schematized representation of situations. The idea is to be able to answer a question, "What will happen if ... ?" by mental simulation. A program could also use some kind of schematized simulation. Its output seems to be propositional. Subjectively, such simulations usually seem to be fragmentary.

11.3 Causal Space and Restricted Situations

Suppose a miner has been started digging in an upper level. If there is nothing that blocks digging, it will break through and fall to the next level down provided nothing happens to interrupt this. I want to use a formalism like that proposed in the draft [McCarthy 94]

The most straightforward way of saying that the miner will eventually fall through is to say he will if no events occur. This is much too strong a condition. Here's an idea.

1. Introduce a concept of *causal space*. In many of the approximate theories we will want to use, causal space will not correspond to real space.
2. Causal space has points. Events occur at points. Points persist from situation to situation. Thus we can talk about the same point in related situations.
3. Distance in causal space is defined by the time of propagation of effects of events. Events cannot affect fluents associated with far away points in a short time.
4. Regions are sets of causal points, normally they will be built up from something like open sets in causal space, i.e. if a point is in the "middle" of a region, sufficiently nearby points will also be in the region.
5. The fewer points there are in causal space close to a given point, the more conveniently local computations can be done. We will want to work with spaces without very unlikely causal connections. For example, President Clinton could conceivably wake up at 5am with a sudden desire to telephone John McCarthy, but it is better to ignore the possibility and put him at a much larger causal distance than this possibility allows.

   This suggests regarding causal space as a metric space. Indeed it may be a metric space, but we want to reserve judgment on what if any topology we shall want to use.

11.4 Formalizing Stratagems

It is stated in the *Lemmings* manual, and I observed it many times, that when a bridge builder bumps its head on a ceiling, it reverses course. An ordinary walker keeps going. To me this was an annoyance. It meant that to be able to reflect walkers, it was necessary to start bridge building several times. Recently, I found a use for the phenomenon, doubtless a use intended by the game designers. If we catch the ex-builder just after it has reversed, it will build a bridge in the opposite direction and will not be followed by other lemmings. This sometimes permits building a ladder of bridges without other lemmings falling off the partial bridges. After the builder has reached the destination the gaps can be closed, and the mob will follow.

I thought of the stratagem while away from the game.

The possibility of the stratagem is implied by previously known lemming physics. However,

1. The stratagem is formulatable in natural language and in logic and can be communicated from one person to another. We need to figure out how to express such stratagems formally.

2. It is not obvious how a program could be made to search for such stratagems.

11.5 Agre and Chapman

Agre and Chapman ([Agre and Chapman, 1987]) describe a program for playing the game of Pengo. According to that article, Pengo does not require planning. An adequate game is played by a program that reacts reflexively to rather simple patterns in the position. For example, their program Pengi uses terms like

$$\textit{the-block-that-the-block-I-just-kicked-will-collide-with}$$

and propositions like

$$there\text{-}is\text{-}no\text{-}block\text{-}suitable\text{-}for\text{-}kicking\text{-}at\text{-}the\text{-}bee,$$

where the dashes indicate that each of the above two expressions is elementary and encoded by some program.

Since the program is specific to the Pengo game, it doesn't correspond to the human ability to learn the rules and then learn how to play effectively.

A program deciding what to do based on a fixed set of such terms and propositions would not work for *Lemmings.* Thus winning one of the games requires making the very first lemming into a floater and using this quality only after the last lemming has emerged. Discovering this requires that the program do planning. The argument the program has to make is something like this.

> I need to make the first lemming into a blocker, but if I plan to blow him up at the end I won't save all the lemmings—which is required in this game. Therefore, he must be undermined rather than blown up. However, if he is undermined as just a blocker, he will fall to his death when undermined. Therefore, he must be made into a floater before he is made a blocker, i.e. at the very beginning. The very last lemming out of the source must also be made a floater and then a miner.

Actually there is a way out whereby it could be claimed that a reactive strategy will work for *Lemmings*, but I doubt that the advocates of reactive programs would want to take it. Namely, if we put a sufficiently rich mental structure into the external situation, then we could imagine a reactive program operating mainly in the mental part of the space could do anything a planner can do. It would be interesting to try to work out details of this idea.

12 References

References

[Agre and Chapman, 1987] **Agre, Philip (pagre@weber.ucsd.edu) and David Chapman (zvona@sail.stanford.edu)**: "Pengi: An Implementation of a Theory of Activity", AAAI National Conference, 1987. Morgan-Kaufman.

[Hardy, 1937] **Hardy, G. H.** (1937): *A course of pure mathematics*, 7th ed., Cambridge University Press. The quotation is from the preface to the 7th edition.

[McCarthy, 1959] **McCarthy, John (1959)**: "Programs with Common Sense", in *Proceedings of the Teddington Conference on the Mechanization of Thought Processes*, Her Majesty's Stationery Office, London. Reprinted in [McCarthy, 1990].

[McCarthy, 1962] **McCarthy, John (1962)**: "Computer Programs for Checking Mathematical Proofs", *Amer. Math. Soc. Proc. of Symposia in Pure Math.*, Vol. 5.

[McCarthy and Hayes, 1969] **McCarthy, John and P.J. Hayes**: "Some Philosophical Problems from the Standpoint of Artificial Intelligence", in D. Michie (ed), *Machine Intelligence 4*, American Elsevier, New York, NY, 1969. Reprinted in [McCarthy, 1990].

[McCarthy, 1979b] **McCarthy, John (1979b)**: "First Order Theories of Individual Concepts and Propositions", in Michie, Donald (ed.) *Machine Intelligence 9*, (University of Edinburgh Press, Edinburgh). Reprinted in [McCarthy, 1990]..

[McCarthy, 1989] **McCarthy, John (1989)**: "Artificial Intelligence and Logic" in Thomason, Richmond (ed.) *Philosophical Logic and Artificial Intelligence* (Dordrecht ; Kluwer Academic, c1989).

[McCarthy 94] **McCarthy, John (1994)**: "Situation Calculus with Concurrent Events and Narrative", manuscript file, /u/jmc/e93/narrative.tex or with URL http://www-formal.stanford.edu/pub/jmc/narrative.dvi.

[McCarthy, 1990] **McCarthy, John (1990)**: *Formalizing Common Sense*, Ablex, Norwood, New Jersey, 1990.

[Psygnosis 93] “Lemmings Manual” - on-line document included with the game. The game may be purchased from computer stores or by calling the company at 44 (51) 709-5755 in England or 800 458-7794 in the U.S.

[McCarthy, 1994b] **McCarthy, John (1994b)**: “Partial Formalizations and the Lemmings Game”, URL http://www-formal.stanford.edu/pub/jmc/lemmings.dvi. This is a reference to the present article.