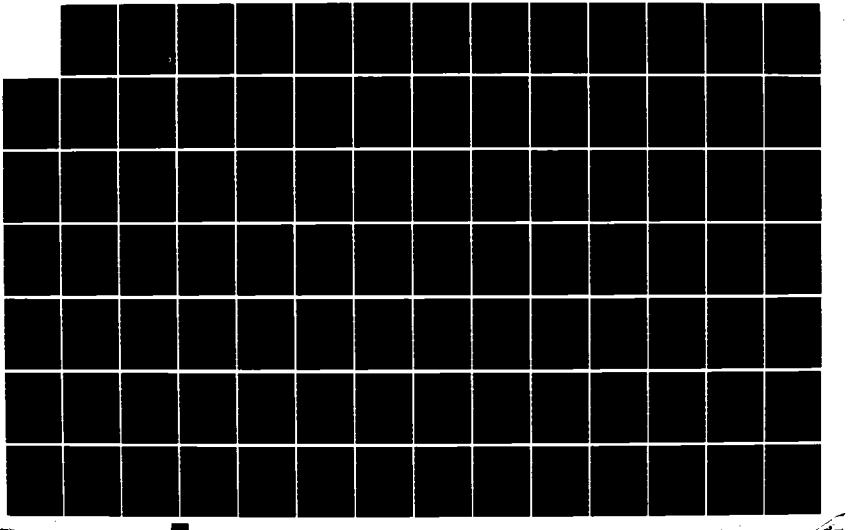
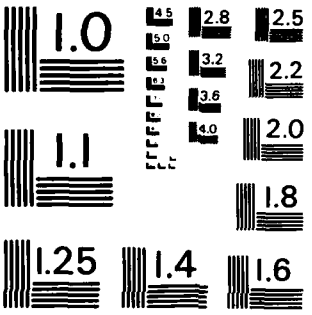
AD-A162 447 TEMPORAL REASONING AND DEFAULT LOGICS(U) YALE UNIV NEW HAVEN CT DEPT OF COMPUTER SCIENCE S HANKS ET AL. OCT 85 YALEU/CSD/RR-430 N00014-85-K-0301 1/2

UNCLASSIFIED F/G 5/10 NL





(12)

TEMPORAL REASONING AND DEFAULT LOGICS

Steve Hanks and Drew McDermott

YALEU/CSD/RR #430

October 1985

YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

TEMPORAL REASONING AND DEFAULT LOGICS

Steve Hanks and Drew McDermott

YALEU/CSD/RR #430

October 1985

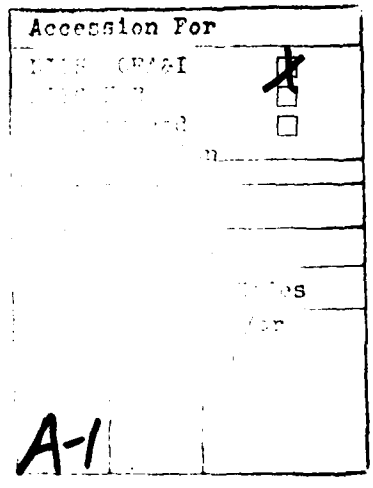
12

TEMPORAL REASONING AND DEFAULT LOGICS

Steve Hanks and Drew McDermott

YALEU/CSD/RR #430

October 1985



This work was supported in part by ONR grant N00014-85-K-0301. Many thanks to Yoav Shoham and Tom Dean for discussions about and comments on this work.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER #430	2. GOVT ACCESSION NO. AD-A163 447	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Temporal Reasoning and Default Logics		5. TYPE OF REPORT & PERIOD COVERED Research Report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Steve Hanks and Drew McDermott		8. CONTRACT OR GRANT NUMBER(s) N00014-85-K-0301
9. PERFORMING ORGANIZATION NAME AND ADDRESS Yale University - Department of Comp. Sci. 10 Hillhouse Avenue New Haven, CT 06520		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, VA 22209		12. REPORT DATE October 1985 13. NUMBER OF PAGES 118
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Office of Naval Research Information Systems Program Arlington, VA 22217		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

temporal reasoning
default reasoning
non-monotonic logic
circumscription

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

We present axioms to represent some simple concepts in temporal reasoning: events occuring at points in time, facts holding true over time, events causing facts to begin, facts causing contradictory facts to cease, and facts tending to remain true unless explicitly forced to cease. To express this last notion we couch the axioms in a default logic: we alternatively consider the logics of McDermott and Doyle (19), of Reiter (23), and of McCarthy (12, 13). We define precisely (through a computer program and its

DD FORM 1 JAN 73 1473

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

formal description) the conclusions we intend be drawn from these axioms, given a particular temporal state of affairs. We prove, however, that these conclusions are not the deductions licensed by any of the above default logics. Further analysis leads us to the conclusion that these logics are inherently incapable of representing this particular kind of default reasoning.

OFFICIAL DISTRIBUTION LIST

Defense Documentation Center
Cameron Station
Alexandria, Virginia 22314

12 copies

Office of Naval Research
Information Systems Program
Code 437
Arlington, Virginia 22217

2 copies

Dr. Judith Daly
Advanced Research Projects Agency
Cybernetics Technology Office
1400 Wilson Boulevard
Arlington, Virginia 22209

3 copies

Office of Naval Research
Branch Office - Boston
495 Summer Street
Boston, Massachusetts 02210

1 copy

Office of Naval Research
Branch Office - Chicago
536 South Clark Street
Chicago, Illinois 60615

1 copy

Office of Naval Research
Branch Office - Pasadena
1030 East Green Street
Pasadena, California 91106

1 copy

Mr. Steven Wong
New York Area Office
715 Broadway - 5th Floor
New York, New York 10003

1 copy

Naval Research Laboratory
Technical Information Division
Code 2627
Washington, D.C. 20375

6 copies

Dr. A.L. Slafkosky
Commandant of the Marine Corps
Code RD-1
Washington, D.C. 20380

1 copy

Office of Naval Research
Code 455
Arlington, Virginia 22217

1 copy

Office of Naval Research
Code 458
Arlington, Virginia 22217
1 copy

Naval Electronics Laboratory Center
Advanced Software Technology Division
Code 5200
San Diego, California 92152
1 copy

Mr. E.H. Gleissner
Naval Ship Research and Development
Computation and Mathematics Department
Bethesda, Maryland 20084
1 copy

Captain Grace M. Hopper, USNR
Naval Data Automation Command, Code 00H
Washington Navy Yard
Washington, D.C. 20374
1 copy

Dr. Robert Engelmore
Advanced Research Project Agency
Information Processing Techniques
1400 Wilson Boulevard
Arlington, Virginia 22209
2 copies

Professor Omar Wing
Columbia University in the City of New York
Department of Electrical Engineering and
Computer Science
New York, New York 10027
1 copy

Office of Naval Research
Assistant Chief for Technology
Code 200
Arlington, Virginia 22217
1 copy

Computer Systems Management, Inc.
1300 Wilson Boulevard, Suite 102
Arlington, Virginia 22209
5 copies

Ms. Robin Dillard
Naval Ocean Systems Center
C2 Information Processing Branch (Code 8242)
271 Catalina Boulevard
San Diego, California 92152
1 copy

Dr. William Woods
BBN
50 Moulton Street
Cambridge, MA 02138
1 copy

Professor Van Dam — 1 copy
Dept. of Computer Science
Brown University
Providence, RI 02912

Professor Eugene Charniak — 1 copy
Dept. of Computer Science
Brown University
Providence, RI 02912

Professor Robert Wilensky — 1 copy
Univ. of California
Elec. Engr. and Computer Science
Berkeley, CA 94707

Professor Allen Newell — 1 copy
Dept. of Computer Science
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA 15213

Professor David Waltz — 1 copy
Univ. of Ill at Urbana-Champaign
Coordinated Science Lab
Urbana, IL 61801

Professor Patrick Winston — 1 copy
MIT
545 Technology Square
Cambridge, MA 02139

Professor Marvin Minsky — 1 copy
MIT
545 Technology Square
Cambridge, MA 02139

Professor Negroponte — 1 copy
MIT
545 Technology Square
Cambridge, MA 02139

Professor Jerome Feldman — 1 copy
Univ. of Rochester
Dept. of Computer Science
Rochester, NY 14627

Dr. Nils Nilsson — 1 copy
Stanford Research Institute
Menlo Park, CA 94025

Dr. Alan Meyrowitz — 1 copy
Office of Naval Research
Code 437
800 N. Quincy Street
Arlington, VA 22217

Dr. Edward Shortliffe — 1 copy
Stanford University
MYCIN Project TC-117
Stanford Univ. Medical Center
Stanford, CA 94305

Dr. Douglas Lenat — 1 copy
Stanford University
Computer Science Department
Stanford, CA 94305

Dr. M.C. Harrison — 1 copy
Courant Institute Mathematical Science
New York University
New York, NY 10012

Dr. Morgan — 1 copy
University of Pennsylvania
Dept. of Computer Science & Info. Sci.
Philadelphia, PA 19104

Mr. Fred M. Griffee — 1 copy
Technical Advisor C3 Division
Marine Corps Development
and Education Command
Quantico, VA 22134

Abstract

We present axioms to represent some simple concepts in temporal reasoning: events occuring at points in time, facts holding true over time, events causing facts to begin, facts causing contradictory facts to cease, and facts tending to remain true unless explicitly forced to cease. To express this last notion we couch the axioms in a default logic: we alternatively consider the logics of McDermott and Doyle [19], of Reiter [23], and of McCarthy [12,13]. We define precisely (through a computer program and its formal description) the conclusions we *intend* be drawn from these axioms, given a particular temporal state of affairs. We prove, however, that these conclusions are *not* the deductions licensed by any of the above default logics. Further analysis leads us to the conclusion that these logics are inherently incapable of representing this particular kind of default reasoning.

Contents

List of Figures

1 Introduction

It has long been recognized that reasoning under conditions of incomplete information is an important part of human cognition. In medical diagnosis, the observed (observable) symptoms rarely allow one to infer the disease; many details are omitted in a story, yet readers are able to supply the missing facts in understanding it. Problem solvers are rarely told that they know all facts relevant to the problem; they have to jump to that conclusion themselves.

Reasoning of this sort often involves "jumping to conclusions": holding a particular belief based on knowledge that a certain state of affairs is *typically* the case, and on *lack* of reason to believe that the observed case is atypical. An overused example of such default reasoning says that given my belief that birds typically fly, and that Tweety is a bird, and *lacking* any direct evidence that Tweety cannot fly, I should jump to the conclusion that Tweety can indeed fly. I may later have to retract that conclusion if somebody tells me that Tweety has a broken wing, is an ostrich, is dead, or has any other condition that might prevent him from flying.

Several formal systems have been proposed to represent explicitly this process of default reasoning. Generally they involve the attempt to formalize the notion of "lacking evidence to the contrary", and are usually referred to as "default", or "nonmonotonic" logics. (Nonmonotonicity refers to the property of these logics that a wff may *cease* to be entailed by a theory as a result of adding additional axioms to it.) We will consider three of these systems: the nonmonotonic logic of McDermott and Doyle [19] and McDermott [15] (hereafter NML), Reiter's default logic [23] (hereafter DL), and McCarthy's circumscription [12,13].

Also see Moore [21].

All of these systems have been subjected to mathematical scrutiny (most notably by Davis [2]), and have been shown to produce anomalous results in some cases. But the examples that break these logics always seem to have an artificial, or even perverse, nature to them. Davis's demonstration that circumscription is not complete rests on an example and technical result from number theory, and the example of a NML theory that has no fixed point involves a default rule expressing "if it's consistent to believe that Tweety can fly, then conclude that Tweety *cannot* fly." On the other hand, the papers in which the logics are introduced ([12,19,23]) use examples that are simple to the point of being of pedagogical but not of practical interest.

So while the negative results may convince us that the systems don't work in all cases, and the positive examples may convince us that they do work in certain very simple cases, we're left wondering about what might happen if we expressed in these logics a practical, reasonably complex problem involving default reasoning. Exactly what conclusions would or would not be licensed by these logics as applied to a practical problem? There are precious few efforts toward answering questions such as these, though a notable exception is the work of Etherington and Reiter [7,8] in formalizing in a default logic a system of inheritence hierarchies with defaults.

Our work begins with a practical problem: representing some simple concepts in temporal reasoning. We develop axioms for expressing the notion of events happening in time and facts holding true over time intervals, and in doing so recognize the need for jumping to certain conclusions. Thus we express our axioms alternatively in the three logics noted above.

We have a clear idea of what we *intend* to express through these axioms (*i.e.* what deductions *should* be licensed by the logics), and provide a computer program that formally expresses our intentions. The question, then, is which (if any) of the logics permit exactly the deductions corresponding to the conclusions drawn by the program. The result is negative: *none* of the logics correctly mirror the program. We examine in some detail the reasons for this failure, trying to ascertain what properties of the problem domain cannot be represented by these nonmonotonic logics, and conclude that all three nonmonotonic logics are inherently incapable of representing the sort of default inference necessary for temporal reasoning.

1.1 Notation

For logical formulas we will use a LISP-like notation. To signify application of a predicate *p* to an individual *x* we will write *(p x)*, and we will use *names* of logical connectives instead of the usual symbols, for example:

(and (p x) (not (q y))).

We use "*if*" to mean implication and "*iff*" to mean bidirectional implication. Syntactic individuals (constant symbols) we will write in capital letters. Variable, predicate, and function names will appear in lower-case letters, but variable names may begin with a question mark; formulas are implicitly quantified universally over all those variables whose names begin with a question mark. Functions will be written in "usual" functional notation (*e.g. f(X)*) to distinguish them from predicates. So we might write

```
(iff (married ?x ?y)
   (and (= ?x spouse(?y))
        (= ?y spouse(?x))))
```

instead of the more conventional

$$\forall x\, y(\mathit{Married}\ x\ y \leftrightarrow x = \mathit{spouse}\ y\ \wedge\ y = \mathit{spouse}\ x).$$

Throughout the paper we'll need to make clear the distinction between syntactic and semantic objects, and between object- and meta-language statements, and we will use typefaces to do so. Generally an italicized sans-serif font will signify a formula in the object language—*(composer BEETHOVEN)*, for example. Semantic individuals and relations will appear in a typewriter-like font, so we might say about a particular model that BEETHOVEN $\in$ composer. Meta-linguistic objects, like sets of formulas or names for (default-logic) extensions, we will set in bold-faced type (*e.g.* **W** may stand for a set of wffs).

2 Formal Systems for Default Reasoning: Semantics and Entailment

To motivate our discussion of the default reasoning systems, we should first give a quick characterization of the temporal domain we set out to represent, explaining how default inference creeps in, and how to represent it in the above default logics. We basically want to reason about events happening at points in time, and about facts holding true over intervals of time. We will be using a point-based temporal logic in the style of McDermott [16]. (This just means that we assume in this paper that events will occur at a time point (instant), and that time intervals will be defined in terms of its begin and end points. Points may be ordered in time: two points can happen at the same time, or one may occur before the other.)

There are four things we particularly want to express:

1. That an event can happen at an instant in time. This might correspond to flipping on a light switch, for example.
2. That given the proper justification, the belief that an particular event has occured may cause us to begin believing a particular fact. (Flipping a light switch may cause a light to become and remain lit, if some preconditions, like the bulb not being burned out, are met.)
3. We stop believing facts when faced with contrary evidence (contradictory facts). If at some later time I come to believe that the same light is unlit (perhaps because I've flipped the switch off) I will believe that the fact that the light is lit has ceased to be true. This is the case of a fact *clipping* another fact.
4. Facts tend to endure over time, unless we have reason to believe otherwise. Now that I believe that the light is unlit I will believe it remains unlit, until I get evidence to the contrary. (This is a simplification of McDermott's [16] notion of *persistence*).

Figure 1 is a graphical representation of the events and facts expressed above. It shows

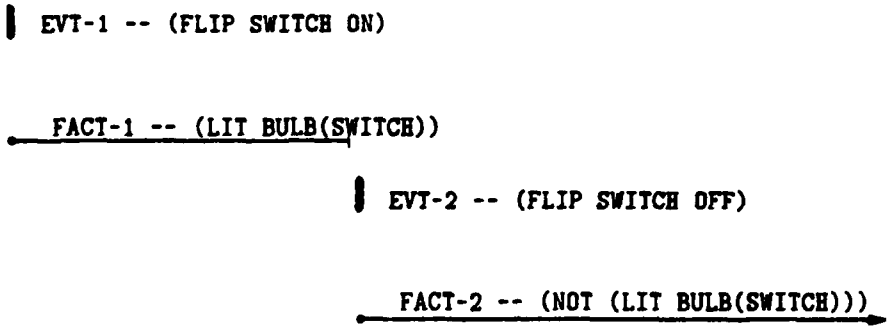


Figure 1: Simple events, persistence, and clipping

---

a switch being flipped ON (an event), which causes a bulb to be lit (a fact). Some time later (farther right in the picture) another event occurs—the switch is flipped to the OFF position—causing a fact asserting that the bulb is not lit. This second fact clips the first, and the *NOT LIT* fact persists indefinitely (as represented by the right arrow). Horizontal position on the page indicates position in time: *EVT-1* and the beginning of *FACT-1* happen simultaneously, the end of *FACT-1* happens after the beginning of *FACT-1* but before *EVT-2*, and the end of *FACT-2* occurs arbitrarily far in the future.

Note that a couple of things have been left implicit: that flipping a switch ON causes the corresponding light to be lit (a statement about causality), that flipping a switch OFF causes the corresponding light to be unlit, and that a bulb can't both be lit and unlit at the same time (a statement about contradiction). Our logic will have to offer a way for us to make these assertions explicit. But for the moment we want only to introduce the graphic

representation of time, and to give a quick introduction to the sorts of things we want to express in the logic.

The notion of facts enduring over time—which we'll sometimes call *persistence*—is the problematic one. The words "lacking evidence to the contrary" imply a default inference. And the default inference is exactly this: for any fact *f*, conclude that it's *not* clipped unless there's specific evidence that it *is*. More formally, if *clipped* is a predicate in our logic and *f* is a fact, we want to believe *(not (clipped f))* by default. We now look at how to express this notion in the three default logics, and where the problems with semantics and entailment arise.

2.1 McDermott's nonmonotonic logic

The nonmonotonic logic of McDermott and Doyle [19] and McDermott [15] is first of all a syntactic extension to the first-order predicate calculus. That is, they extend the language of FOPC to include a modal operator *M*, that is supposed to mirror the notion of provability within the logic. That is, *(M P)* should be a theorem just in case it's "consistent to believe P", which might be more formally expressed as a rule of inference:

$$\nvdash (not\ P) \Rightarrow \vdash (M\ P).$$

("From the inability to deduce *(not P)* conclude *(M P)*.") The problem is that this definition is directly circular: a system's rules of inference themselves define what's derivable in the system, so the notion of derivability can't be used to define an inference rule.

Instead, nonmonotonic derivability is defined in terms of fixed points of the operator

NM, defined as follows:

$$NM_{\mathbf{A}}(S) = Th(\ \mathbf{A} \cup As_{\mathbf{A}}(S)\ )$$

where **A** is a theory in a first-order language $\mathcal{L}$ extended with the operator M, Th is (monotonic) deductive closure, and $As_{\mathbf{A}}$ can be thought of as the set of assumptions that can be "jumped to" from a set of formulas S:

$$As_{\mathbf{A}}(S) = \{(M\ q) : q \in \mathcal{L},\ and\ (not\ q) \notin S\} - Th(\mathbf{A}).$$

The set of theorems (nonmonotonically) derivable from **A** is then defined as the *intersection* of all fixed points of the NM operator:

$$\bigcap \{S : NM_{\mathbf{A}}(S) = S\},$$

or the entire language $\mathcal{L}$ if there is no fixed point.

In general it's not possible to determine how many fixed points a particular theory **A** will have. The desirable case would be a guarantee that **A** has exactly *one* fixed point, especially since the proof procedure provided in McDermott and Doyle [19] and in Doyle [4] answers the question of whether a formula is in *all* fixed points. Thus if a theory has several fixed points, and each in some way describes a separate state of affairs, the proof procedure is of no use to the reasoner trying to maintain a coherent view of the world.

One situation in which the multiple fixed point problem arises is when the theory contains "conflicting default rules". We won't bother attempting a formal characterization of such theories, but the classic example is the following:

```
; all Quakers are pacifists, unless there's reason to believe
; otherwise
(if (and (quaker ?x) (M (pacifist ?x)))
    (pacifist ?x))

; all Republicans are non-pacifists, unless there's reason to
; believe otherwise
(if (and (republican ?x) (M (not (pacifist ?x))))
    (not (pacifist ?x))))

; Nixon is both a Quaker and a Republican
(quaker NIXON)
(republican NIXON)
```

This theory has two fixed points:

(1)	(2)
(quaker NIXON)	*(quaker NIXON)*
(republican NIXON)	*(republican NIXON)*
(pacifist NIXON)	*(not (pacifist NIXON))*
(not (M (not (pacifist NIXON))))	*(M (not (pacifist NIXON)))*
(M (pacifist NIXON))	*(not (M (pacifist NIXON)))*

We have conflicting defaults in that one predicate (*quaker*) applied to an individual leads us to jump to a particular conclusion about that individual, and a second predicate applied to an individual (*republican*) licenses a contradictory conclusion. We get multiple fixed points if there is some individual for whom both predicates hold. Both fixed points describe a particular state of the world, each consistent within itself, but incompatible with the other fixed point. Notice the problem with using a proof procedure that decides whether a particular formula (*e.g.* *(pacifist NIXON)*) is in all fixed points. It would answer "no", but would offer the same verdict for that formula's negation. Clearly we would like to adopt one fixed point and "stick with it", but there's certainly nothing in the logic that should cause us to favor one or the other.

We should note at the outset that the problem of multiple fixed points is mainly a computational one. Different fixed points for a theory often represent different plausible and incomparable states of the world, but states that are all consistent with the default rules. (Such is the case with the "Nixon" example, but later in the paper we will see another theory in which one fixed point describes a world state that is counterintuitive). It's clear however, that the whether this logic is of *practical* value depends on whether we can identify a unique fixed point for a particular theory (or at least verify that *all* fixed points have certain desired properties). If we have just one, the proof procedure works just right (it marks as nonmonotonically derivable exactly those formulas in the unique fixed point). If not, the proof procedure may tell us nothing of value.

2.2 Reiter's default logic

While Reiter's [23] default logic (hereafter "DL") looks much like the one we just discussed, those similarities are fairly superficial. The first thing to note about DL is that it enforces a strong distinction between the (monotonic) first-order wffs and the (nonmonotonic) default rules. Recall that in NML a first-order wff (say *P*), and a wff involving the nonmonotonic operator *M* (say *(M P)*) were afforded the same status in the object language. And what we might consider a default rule, such as

```
(if (and (bird x) (M (fly x)))
    (fly x))
```

is *itself* to be considered just a wff in some nonmonotonic theory.

In DL the set of default rules is described in a meta-language, not in the same language (FOPC) that describes the rest of the world knowledge. So a "default theory" is composed

of a set of ordinary wffs (abbreviated as **W**), along with a set of default rules (**D**). In Reiter's notation a default theory is abbreviated by the symbol Δ, so to summarize, we have

$$\Delta = (\mathbf{W}, \mathbf{D}).$$

Each default rule d looks like this:

$$\frac{\alpha(\vec{x}) \ : \ \mathbf{M}\,\beta(\vec{x})}{\gamma(\vec{x})}$$

where $\vec{x}$ is a vector of variables, and α, β, and γ are wffs whose free variables are limited to those in $\vec{x}$. The rule is intended to mean: for any set of individuals $\vec{k}$, if $\alpha(\vec{k})$ is true, and it's consistent to believe that $\beta(\vec{k})$ is true, then conclude that $\gamma(\vec{k})$ is true. (In Reiter's definition, there can be any (finite) number of β wffs, but all the default rules we'll see involve only one such β.) Notice that the symbol **M** has once again showed up, and that once again it is to be read as "consistent", but here **M** is included for aesthetics only. It is not part of the object language.

The analogue in DL to the notion of a fixed point is that of an *extension*. Every default theory defines zero or more extensions. An extension is itself a set of wffs, a superset of **W**, and is intended to be the set of deductions that can be drawn from **W** along with "licensed" application of the default rules in **D**.

In describing NML we had somewhat of a problem coming up with a precise and satisfying definition of "consistent", and since we defined the default rules using that word, we once again have to make our intended meaning clear. What we mean is that for any extension **E** of a default theory Δ containing a default rule d as above, and for any vector of individuals $\vec{k}$, it should be the case that if $\alpha(\vec{k}) \in \mathbf{E}$, and if *(not* $\beta(\vec{k})$*)* $\notin \mathbf{E}$, then $\gamma(\vec{k}) \in \mathbf{E}$.

The proof procedure Reiter suggests is similar to that of McDermott and Doyle—it answers for some sentence φ whether φ is in *some* extension. And as you might suspect, there's no guarantee that there will be exactly one. Reiter does prove that for a particular class of default theories, called *normal* defaults, that there is *at least one* extension. Normal default theories are those in which all default rules are of the form

$$\frac{\alpha(\overline{x}) \ : \ \mathbf{M}\ \beta(\overline{x})}{\beta(\overline{x})}. \tag{1}$$

(All the defaults of interest to us in this paper are normal, so we will assume from now on that when we speak of a default theory we speak of a normal default theory. We are ignoring one more technical point here, which is that Reiter's results apply to "closed normal default theories"—those in which α, β, and γ are closed wffs. Note that the sample default rule (1) above is free in variables $\overline{x}$, thus is not closed. We will use notation like that in (1) to denote an (infinite) class of closed normal default rules, formed by substituting for all occurences of $\overline{x}$ all constant individuals in the language. Since Reiter's results place no restriction on the number of default rules in a normal default theory, his results still apply.)

We're still faced with the problem of multiple extensions, though. As an example consider again the *NIXON* axioms from the previous section—we can recast it as a default theory, like this:

$$\mathbf{W} = \{\ \textit{(quaker NIXON)},\ \textit{(republican NIXON)}\ \}$$

$$\mathbf{D} = \{\ \frac{\textit{(quaker ?x)} \ : \ M\ \textit{(pacifist ?x)}}{\textit{(pacifist ?x)}}\ ,\ \frac{\textit{(republican ?x)} \ : \ M\ \textit{(not (pacifist ?x))}}{\textit{(not (pacifist ?x))}}\ \}$$

This (normal) default theory has two extensions:

$\mathbf{E}_1$ = { *(quaker NIXON)*, *(republican NIXON)*, *(pacifist NIXON)* }

$\mathbf{E}_2$ = { *(quaker NIXON)*, *(republican NIXON)*, *(not (pacifist NIXON))* }

As above, the proof procedure is of no use to us in this case—it will answer "yes" both to *(pacifist NIXON)*, and to its negation. Once again there's no way to pick one state of the world and stick to it (and besides, it's not clear which extension should be preferred). [1]

Whether there are significant practical differences between the sorts of situations NML and DL can represent, and whether there is a class of problems for which one system will produce a single extension and other will not, is unclear (indeed is one topic addressed by this paper). Obviously the NML syntax is more flexible than that of DL, in that the *M* operator can appear anywhere within a wff. So any default theory can be expressed as an NML theory, but not *vice versa*. On the other hand, Reiter claims that this is of no practical importance, since all realistic problems in default reasoning can be represented by normal defaults anyway.

2.3 Circumscription

McCarthy [12,13] proposes the process of "circumscribing" an axiom in the first-order predicate calculus over a particular predicate (or predicates), as a means of allowing nonmonotonic inference. The intent of circumscribing an axiom over a predicate *P* is that, in the resulting theory, any individual *k* not forced (by the original axiom) to have property P does *not* have property P—that is, after circumscription *(not (P k))* would follow from the circumscribed axiom just if *(P k)* was not a theorem of the original axiom.

[1] Recent work by Etherington ([7]) reports on a proof procedure that will restrict itself to a single extension, even if there is more than one in the corresponding default theory.

There are several formal systems floating around the literature bearing the name "circumscription." In the original paper (McCarthy [12]) the axiom is augmented by a (first-order) axiom schema, thus both the original axiom and the circumscribed axiom are first-order formulas. In that paper is also presented a model-theoretic characterization of what formulas are entailed by the circumscribed axiom (which we will discuss below). In a subsequent paper (McCarthy [13]) appears a restatement of the formalism in which circumscription is effected by augmenting the axiom with a second-order formula. In this later version of circumscription one is permitted to circumscribe over an arbitrary first-order wff instead of over a single predicate, and the concept of allowing predicates to vary as parameters to the circumscription (which we will discuss below) is introduced. The model theory for the second-order theory is not worked out in that paper. Lifschitz [11] provides a further generalization of the notion of circumscription within the second-order framework.

Since we're not overly concerned with the technical details of circumscription, and since our result holds no matter which version is chosen, we will feel free in the discussion that follows to use notation and concepts from several versions of the formalism, as clarity dictates.

Let's start with an axiom **A**—a FOPC formula that contains predicate symbols P, $\overline{Q}$, and $\overline{R}$, where $\overline{Q}= \{Q_1, Q_2, \ldots\}$ and $\overline{R}= \{R_1, R_2, \ldots\}$. (In the "Nixon" example above we wrote a set of axioms that were implicitly conjoined. Note that here we are using a single axiom, but it could well be a conjunction.) The circumscription of **A** with respect to P, using parameters $\overline{Q}$ is the following formula:

(1) *(and* **A***(P.* $\vec{Q}$*.* $\vec{R}$*)*
(2) *(forall (P'.* $\vec{Q}$*')*
(3) *(if (and* **A***(P'.* $\vec{Q}$*'.* $\vec{R}$*)*
(4) *(forall* ($\vec{x}$*) (if (P'* $\vec{x}$*) (P* $\vec{x}$*))))*
(5) *(forall* ($\vec{x}$*) (iff (P'* $\vec{x}$*) (P* $\vec{x}$*))))))*

(We will refer to this formula as "the circumscription of **A** with respect to *P*, using $\vec{Q}$ as parameters", and abbreviate it as Cs(**A**, *P*, $\vec{Q}$). The notation **A**(*P'*, $\vec{Q}$', $\vec{R}$) stands for the axiom **A** with the syntactic substitution of *P'* for *P*, Q_1' for Q_1, Q_2' for Q_2, ..., etc. Note that the first conjunct in Cs(**A**, *P*, $\vec{Q}$) is **A** itself, so any formula entailed by **A** will also be entailed by the circumscription of **A**.)

What the formula means is that for any choice of substitute predicate *P'* for *P*, if *P'* satisfies the original axiom **A**, and if *P'* is at least as strong as *P*, then *P* and *P'* hold of exactly the same individuals. Another way to put it is that the formula "selects out" the strongest *P'* that satisfies **A**. (All subject to variation in the parameter predicates $\vec{Q}$—we will see what this means in a moment.)

As an example, let's take **A** to be the conjunction of the following formulas:

(a') *(if (olive ?x) (green ?x))*
(b') *(if (frog ?x) (green ?x))*
(c') *(olive MAX)*
(d') *(green BLOCK-35).*

We intend that circumscribing over *green* pick out just the green individuals. That is, we would expect to be able to deduce *(green MAX)*, and *(green BLOCK-35)*, but for any individual *Y* other than those two, we would expect to be able to deduce *(not (green Y))*. We would thus expect the circumscribed axiom to entail the formula

(6) (iff (or (= ?x MAX) (= ?x BLOCK-35))
(green ?x))).

(Note that (6) is an instance of the consequent of the circumscription formula, (5) above. In other words, we seek an equivalence between *green* and our choice of *green'*, where *green'* is the disjunction in the first line of (6).)

To deduce (6) from the circumscription axiom we have to do two things: demonstrate both that our *green'* can satisfactorily be substituted into the original axiom (thus satisfying line (3) above), and that *green'* is at least as strong as *green* (satisfying line (4)). The latter task is easy, in that involves verifying the validity of the formula

(7) (if (or (= ?x MAX) (= ?x BLOCK-35))
(green ?x)))

which is obviously true. Verifying (3) involves substituting our version of *green'* into all formulas in **A** that mention *green*, yielding

(a') (if (olive ?x)
(or (= ?x MAX) (= ?x BLOCK-35)))
(b') (if (frog ?x)
(or (= ?x MAX) (= ?x BLOCK-35)))
(d') (or (= BLOCK-35 MAX) (= BLOCK-35 BLOCK-35)).

It turns out that we can't verify (a') or (b'), conceptually because it requires the information that there are no olives other than MAX, and no frogs at all—information that's not contained in **A**. So far we've allowed no predicates to vary as parameters, so we must conclude that the desired formula (6) is *not* entailed by Cs(**A**, *green*, ϕ).

What we must realize is that the definition of *green* in some sense *depends* on the definitions of *olive* and *frog*, so when we circumscribe over *green* we must allow *olive* and *frog* to

vary as parameters. In doing so we note that for the following definitions

(iff (olive' ?x) (= ?x MAX))
(iff (frog' ?x) FALSE)

the substitutions into **A** now work right:

(a") *(if (= ?x MAX)*
(or (= ?x MAX) (= ?x BLOCK-35)))
(b") *(if FALSE*
(or (= ?x MAX) (= ?x BLOCK-35)))
(c") *(= MAX MAX)*
(d") *(or (= BLOCK-35 MAX) (= BLOCK-35 BLOCK-35)).*

and since (7) is valid as before, the precondition formulas (3) and (4) are satisfied, and the equivalence (6) is entailed by the circumscribed axiom. In other words, Cs(**A**, *green*, {*olive*, *frog*}) entails formula (6).

Two important questions remain unanswered for this method of circumscription: how does one decide which predicates to use as parameters when one undertakes a particular circumscription? And what does it *mean* when one decides to include or exclude a particular predicate as a parameter? There are no satisfying answers at this point. (See, for example, McCarthy [13, sect. 5]). We will ignore for the rest of this paper the problem of choosing parametric predicates, and just choose whichever prove convenient for expository reasons.

2.4 Model theory for circumscription

To get a precise characterization of what formulas are entailed by the circumscribed axiom, McCarthy (in [12]) provides some model-theoretic results. Perlis and Minker [22] extend these results to the version of circumscription allowing predicates to vary as parameters (but

without resorting to McCarthy's second-order formalism). The discussion below summarizes the model theory of circumscription as presented in [22].

Since the intent of circumscription is to "minimize" the individuals for which $\mathbf{A} \vdash_{(P, \overline{Q})} (P\ x)$[2], the counterpart of an extension or a fixed point will be that of a model of **A** *minimal* in P, subject to variation in $\overline{Q}$. Let $\mathcal{M}$ and $\mathcal{N}$ be models of **A**. We say that $\mathcal{M}$ is a submodel of $\mathcal{N}$ in P relative to $\overline{Q}$ if $\mathcal{M}$ and $\mathcal{N}$ have the same domain, if they agree on all predicates in **A** besides P and $\overline{Q}$, and if the extension of P in $\mathcal{M}$ is a proper subset of the extension of P in $\mathcal{N}$. $\mathcal{M}$ is minimal in P (relative to $\overline{Q}$) if there are no models, except for $\mathcal{M}$ itself, that are submodels of $\mathcal{M}$ in P (again relative to $\overline{Q}$). We say that a sentence φ is *minimally entailed* by **A** with respect to P and relative to $\overline{Q}$ (abbreviated $\mathbf{A} \models_{(P, \overline{Q})} \varphi$) if φ is true in all models of **A** minimal in P (relative to $\overline{Q}$).

McCarthy proves for circumscription a result analoguous to the soundness of FOPC: if $\mathbf{A} \vdash_{(P,\overline{Q})} \varphi$ then $\mathbf{A} \models_{(P,\overline{Q})} \varphi$. But the converse, the completeness result, does not hold. Davis [2] provides a counterexample for the general case, but Perlis and Minker [22] show that completeness *does* hold in some special cases—for example when the extensions of P and $\overline{Q}$ are all finite.

So how can we characterize the formulas the formulas entailed by Cs(**A**,P, $\overline{Q}$)? Well, *at best* we're in the same position with circumscription as we were with NML or DL: even assuming completeness, we can get a proof of a sentence φ from Cs(**A**,P, $\overline{Q}$) just in case φ is true in *all* models of **A** minimal in P. And if there is more than one minimal model for a theory, and if a sentence φ is *not* true in all of them, then the contrapositive of soundness

[2]We will use $\mathbf{A} \vdash_{(P, \overline{Q})} \varphi$ to abbreviate Cs(**A**, P, $\overline{Q}$) $\vdash \varphi$, and when the choice of $\overline{Q}$ is unimportant we may neglect to mention the parameter predicates, and say $\mathbf{A} \vdash_P \varphi$.

tells us that φ does *not* follow from the circumscribed axiom.

So the question arises again: for any given **A**, are we assured of *any* minimal models, or, better yet, of exactly one? The answer (to nobody's surprise) is *no*. Minimal models do not always exist (Davis [2]), and we can once more employ our "Nixon" example to show that there may be more than one.

The Nixon example doesn't immediately translate into a form suitable for circumscription, but we can use a trick explained in McCarthy [13] and McDermott [15] to obtain the following:

```
(if (and (quaker ?x)
         (not (ab aspect1(?x))))
    (pacifist ?x))

(if (and (republican ?x)
         (not (ab aspect2(?x))))
    (not (pacifist ?x)))


(quaker NIXON)
(republican NIXON)
```

where *ab* is a predicate in some sense representing "abnormality." The first formula says that for any individual *?x*, if *?x* is a Quaker and *?x* is not abnormal in aspect1, then *?x* is a pacifist. So we can now circumscribe these formulas with respect to *ab* (using *pacifist*, *quaker* and *republican* as parameters).

Note that the extension of *ab* in any model of **A** must contain either `aspect1(NIXON)` or `aspect2(NIXON)`—otherwise it would follow from **A** both that `NIXON` $\in$ `pacifist`, and `NIXON` $\notin$ `pacifist`. Furthermore, the extension of *quaker* and of *republican* in any model must both contain the individual *NIXON*.

Now consider two models, $\mathcal{M}_1$ and $\mathcal{M}_2$, where $\mathcal{M}_1$ has $\mathbf{ab}_1 = \{\mathtt{aspect1(NIXON)}\}$ as its extension of *ab*; analogously, $\mathbf{ab}_2 = \{\mathtt{aspect2(NIXON)}\}$. As we noted above, ***any*** model of the axioms must include one or the other of these individuals in its extension of *ab*, so there can be no submodels in *ab* either of $\mathcal{M}_1$ or of $\mathcal{M}_2$. Therefore both $\mathcal{M}_1$ and $\mathcal{M}_2$ are minimal in *ab*. But since $\mathtt{NIXON} \in \mathbf{pacifist}_1$ and $\mathtt{NIXON} \notin \mathbf{pacifist}_2$, neither *(pacifist NIXON)* nor its negation follows from the circumscribed axioms.

2.5 Summary and implications

In order to represent the idea of persistences (facts) enduring over time, we are forced to use a default logic. Any of the three systems we just examined seem adequate to represent this idea: a single axiom involving *M* in NML, a single (normal) default rule in DL, and circumscribing over *clipped* in circumscription.

Despite their dissimilar appearance, all three systems generally seem to fall prey to the same sorts of problems: whether it be a NML fixed point, a DL extension, or a minimal model in circumscription, we need a unique one in order to guarantee that we can make coherent deductions from the theory. No general result (or any but the most trivial results, for that matter) tells us when we might expect to get a unique fixed point, although we have seen that the case of conflicting default rules (as illustrated by the Nixon example) will tend to screw up all three logics. (By that I mean "computationally screw up." As mentioned above, the existence of multiple fixed points or extensions may indeed be the appropriate way for the logics to behave under these circumstances, but without the means to evaluate the theory to see how many fixed points we have, to examine each fixed point, then to choose the one most appropriate to our reasoning task, it's not clear that these

theories are of any practical value.)

But for the case of temporal reasoning (where the desired nonmonotonic inference is "non-clipping by default") we have to wonder whether the pessimistic general results should really hinder us. After all, our temporal problem doesn't involve the use of conflicting default rules as we saw them above; as a matter of fact there's only *one* default rule, and it's the most reasonable kind—saying "if it's consistent to believe *not clipped*, then believe it." Perhaps since we are so controlled in our use of the default rules, we can expect better behavior from the three logics. The point of the rest of the paper will be to find out.

3 Representing Time in Logic

Before getting on to a detailed look at the axioms for reasoning about time, we should first review the four things we want to represent in the simple case we'll be concerned with throughout the paper. Recall the things we wanted to express:

1. events happen at particular times (instants)
2. facts hold true over intervals of time
3. the occurrence of events may cause facts to begin
4. beginning to believe a particular fact may cause us to stop believing a contradictory fact
5. lacking evidence to the contrary, facts tend to endure (persist) over time.

Now we should say right off that in no way do we consider this the definitive temporal representation. To the contrary, our aim throughout the research was to make the particular representation problem as simple as possible, while still maintaining the essence of the problem (the ability to represent the five things above). Some of the most obvious simplifying assumptions we have made are: that events happen instantaneously, that when an event causes a fact it does so immediately, and that facts tend to endure forever, rather than for some finite lifetime dependent on the type of fact. These assumptions have allowed us to do away with metric considerations altogether—there are no distances between time points or durations to time intervals, only ordering information. (See McDermott [16] for how these extensions might be accomplished.) Relaxing these assumptions to provide a richer temporal theory would only tend to worsen the technical problems described below, and would obscure the discussion.

3.1 Problem-independent axioms

We admit three types of individuals into the logic: (time) *points*, *patterns*, and *tokens*. Points mark an instant in time. An event occurs at a particular point, and facts hold over intervals that are defined by their begin and end points. Points may be ordered with respect to other points.

Patterns describe *what* happens when an event occurs (*e.g.* ***(DROP JOHN VASE)***), or *what* is asserted to be true during the duration of a fact (*e.g.* ***(SAD JOHN)***). While patterns look like ordinary predicate calculus sentences (***JOHN*** being an individual and ***SAD*** being a predicate applied to it), this is not the case. Patterns are terms, so, for example ***(SAD JOHN)*** and ***(NOT (SAD JOHN))*** are just two terms and are not inherently contradictory (because ***NOT*** in this case is not logical negation). We ignore the way these patterns might be combined—how *and* *or* and *not* might be handled, for example. (See Allen [1], Moore [20] and Shoham [25] for discussion.) For our purposes, the only things that patterns can do is *contradict* each other, and we will have to supply explicit axioms saying when this does and does not happen.

Tokens are just unique identifiers. Every occurrence of an event or a fact has a unique token associated with it, as will a couple of other things to be explained.

The following predicates operate on points, patterns, and tokens: *pcause*, *event*, *persist*, *point*, *contradict*, *clipped*, $\sim$, $\prec$, $\preceq$. Here are their intended interpretations:

- ***(pcause tok ofact-pat event-pat nfact-pat)***

  Using *pcause* we make explicit the causal forces at work within the system. If an event with the pattern *event-pat* occurs while a fact with the pattern *ofact-pat* is true,

this causes a new fact, with pattern *nfact-pat*—its begin point will coincide with the occurrence of the event.

- *(event tok pat pt)*

  Tok identifies the occurrence of an event, asserting pattern *pat*, that happens at point *pt*.

- *(persist tok pat bp ep)*

  Tok identifies the occurrence of a fact (persistence), asserting pattern *pat*, and holding true over the interval beginning with point *bp* and ending with point *ep*. *Bp* must occur before *ep*.

- *(point pt)*

  Individual *pt* is a point.

- *(contradict pat1 pat2)*

  Pattern *pat1* contradicts *pat2*. Facts with patterns that contradict each other tend to clip each other.

- *(clipped tok pt)*

  Tok is a token denoting a persistence, *pt* is a point, and *pt* "clips" *tok*. It follows that *pt* occurs after the end point of *tok*. The begin point of a persistence clips a contradictory persistence token, and every point occurring later than that point clips the token as well.

- *(~ p1 p2).*

  (≺ p1 p2).

  (≼ p1 p2)

These predicates serve to order points in time. The first means that *p1* and *p2* happen simultaneously, the second says that *p1* happens before *p2*. The third *generally* means that *p1* happens at the same time as or before *p2*, but if both *p1* and *p2* are persistence end points it may be the case that both *(≼ p1 p2)* and *(≼ p2 p1)* hold, but that *(~ p1 p2)* does not. In that case ≼ can be taken to mean that the points can occur in either order.

The above predicates will hereafter be known as the "temporal predicates."

Next we consider a set of axioms—mostly implications that express formally the predicates' definitions as they were explained above. These axioms are independent of any particular temporal situation we want to represent, and as a group will be called the "temporal axioms", or the set **T**. We will skip some of the less interesting ones; Appendix A provides a complete list.

The first axiom formalizes the relation between *pcause*, *event*, and *persist*—it's the way we can infer the existence of new persistences, and is just a formal restatement of the definition of *pcause* above. (This is axiom 1 in appendix A.)

```
(if (and (pcause ?pc-tok ?ofact-pat ?evt-pat ?nfact-pat)
         (persist ?ofact-tok ?ofact-pat ?ofact-bp ?ofact-ep)
         (event ?evt-tok ?evt-pat ?evt-pt)
         (≼ ?ofact-bp ?evt-pt)
         (≼ ?evt-pt ?ofact-ep)))
    (and (persist
            pt(?pc-tok. ?evt-tok. ?ofact-tok)
            ?nfact-pat
            pb(?pc-tok. ?evt-tok. ?ofact-tok)
            pe(?pc-tok. ?evt-tok. ?ofact-tok))
         (~ pb(?pc-tok. ?evt-tok. ?ofact-tok) ?evt-pt)))
```

Note that in order to *get* a persistence from this axiom, we must first *have* a persistence.

So how do we get the first one? We explicitly assert the existence of a persistence asserting the pattern *ALWAYS*. The begin point of this fact occurs before every other point, and the fact is clipped by no point:

```
(persist PTA ALWAYS PBA PEA)

(if (point ?p)
    (or (= ?p PBA)
        (≺ PBA ?p))

(if (point ?p)
    (not (clipped PTA ?p)))
```

Next we formalize the clipping process: a point *p* can clip a fact named by token *tok* if *p* is the begin point of a token *tok'* that is contradictory to *tok*, and if *tok* begins before *tok'* does. A point *p* can also clip a fact denoted by *tok* if it falls after a point *p'* that clips *tok*:

```
(if (and (persist ?fact-tok ?fact-pat ?fact-bp ?fact-ep)
         (persist ?clip-tok ?clip-pat ?clip-bp ?clip-ep)
         (contradict ?fact-pat ?clip-pat)
         (≺ ?fact-bp ?clip-bp))
    (clipped ?fact-tok ?clip-bp))

(if (and (persist ?fact-tok ?fact-pat ?fact-bp ?fact-ep)
         (clipped ?fact-tok ?p1)
         (≤ ?p1 ?p2))
    (clipped ?fact-tok ?p2))
```

Next the relation between clipping and point ordering: if a point clips a token, the point follows (in time) the token's end point. If a point does *not* clip a token, the end point of that token occurs after the point.

```
(if (and (persist ?tok ?pat ?bp ?p1)
         (clipped ?tok ?p2))
    (≺ ?p1 ?p2))
```

```
(if (and (persist ?fact-tok ?fact-pat ?fact-bp ?fact-ep)
         (point ?p)
         (not (clipped ?fact-tok ?p)))
    (≼ ?p ?fact-ep))
```

Note that in the first case we use $\prec$ and in the second we use $\preceq$. We need to do that to handle correctly the ordering between persistence end points. Consider, for example, persistences tok_1 and tok_2, with end points p_1 and p_2 respectively, and assume that *(not (clip* tok_1 p_2*))* and *(not (clip* tok_2 p_1*))*. Axiom 6 forces the conclusions *(*$\preceq$ p_1 p_2*)* and *(*$\preceq$ p_2 p_1*)*, while it needn't be the case that *(*$\sim$ p_1 p_2*)*.

The begin point of any persistence must occur before its end point. Persistence begin and end points are *point*s, as are event points:

```
(if (persist ?tok ?pat ?p1 ?p2)
    (≼ ?p1 ?p2))

(if (persist ?tok ?pat ?p1 ?p2)
    (and (point ?p1)
         (point ?p2)))

(if (event ?tok ?pat ?p)
    (point ?p))
```

Contradiction is symmetric:

```
(if (contradict ?pat1 ?pat2)
    (contradict ?pat2 ?pat1))
```

We'll skip the rest of the axioms, referring the interested reader to Appendix A—all that's left are axioms defining the point-ordering predicates. They say things like $\sim$ is reflexive, symmetric and transitive, that substituting points that are coincident preserves the relations $\prec$ and $\preceq$, that if two points are $\sim$ they are also $\preceq$, and so on.

3.2 The default rule

It may well occur to the reader that we haven't needed to use any default rules yet. Furthermore, given these axioms it's not clear exactly why we need one at all. To see why, and where, refer to axiom 6, which says if a point does *not* clip a token, then the token's end point falls after that point. That's pretty much what we wanted to express when we said that facts endure over time. But more precisely, that axiom says that if we can deduce *(not (clipped ?tok ?p))* then we can also deduce *($\preceq$?p end(?tok))*. But in general the axioms don't give us any way to deduce *(not (clipped ...))*—in fact, that's exactly the conclusion we need to jump to.

We needn't commit ourselves to a particular logic at this point; the default inference is easily expressed in any of the three. In NML we would simply add the axiom

```
(if (and (persist ?tok ?pat ?bp ?ep)
         (point ?p)
         (M (not (clipped ?tok ?p))))
    (not (clipped ?tok ?p)))
```

In Reiter's logic the above axioms make up part of the set **W** (problem-specific axioms, below, complete the set), and the set **D** consists of a single class of default rules defined by:

$$\frac{(and\ (persist\ ?tok\ ?pat\ ?bp\ ?ep)\ (point\ ?p))\ :\ \mathbf{M}\ (not\ (clipped\ ?tok\ ?p))}{(not\ (clipped\ ?tok\ ?p))}$$

To represent this default inference using circumscription we just circumscribe the axioms over the predicate *clipped*, letting all the other temporal predicates vary as parameters in the circumscription.

Hereafter I'll talk about the set **T** as if it has been augmented with the appropriate

default rule. For the most part the discussion will make clear which logic we're adopting for the moment, or it won't matter which one.

3.3 Problem-specific axioms

We've now seen the problem-independent axioms, **T**. For the most part these axioms were implications—they told us what would happen, for example, *if* we had a particular *pcause* assertion, and a particular *event* assertion, and a particular *persist* assertion, *etc.* But so far we have no events, no causality, no contradiction. So no persistences (other than *ALWAYS*), no clippings, and no points.

When one wants to reason about a particular temporal state of affairs one must supply assertions postulating the existence of *events*, of *pcauses*, and of *contradictions*. We will assume that there are a finite number of each; *in the next section we will make more* stringent restrictions on the form they may take. Consider figure 2, which contains a sample set. We will use this sample problem through the rest of the paper.

The *pcause* assertions say the following:

- if a person is born, then it's always the case that that person will start being alive
- if someone loads a gun, then it's always the case that the gun will start being loaded
- if someone is shot with a gun, and the gun is loaded, then that person will start being dead
- if someone is alive, and that person wins the sweepstakes, then that person will start being rich.

The *contradict* assertions say that at no point in time can a person be both alive and dead. In other words, a fact asserting *(DEAD X)* will clip a fact asserting *(ALIVE X)*, and *vice versa*.

```
(pcause PC-1 ALWAYS (BORN ?x) (ALIVE ?x))

(pcause PC-2 ALWAYS (LOAD ?x) (LOADED ?x))

(pcause PC-3 (LOADED ?g) (SHOOT ?x ?g) (DEAD ?x))

(pcause PC-4 (ALIVE ?x) (WIN-SWEEPSTAKES ?x) (RICH ?x))

(event EVT-1 (BORN JOHN) PEVT-1)

(event EVT-2 (LOAD GUN) PEVT-2)

(event EVT-3 (SHOOT JOHN GUN) PEVT-3)

(event EVT-4 (WIN-SWEEPSTAKES JOHN) PEVT-4)

(≺ PEVT-1 PEVT-2)

(≺ PEVT-2 PEVT-3)

(≺ PEVT-3 PEVT-4)

(contradict (DEAD ?x) (ALIVE ?x))
```

Figure 2: Sample problem axioms

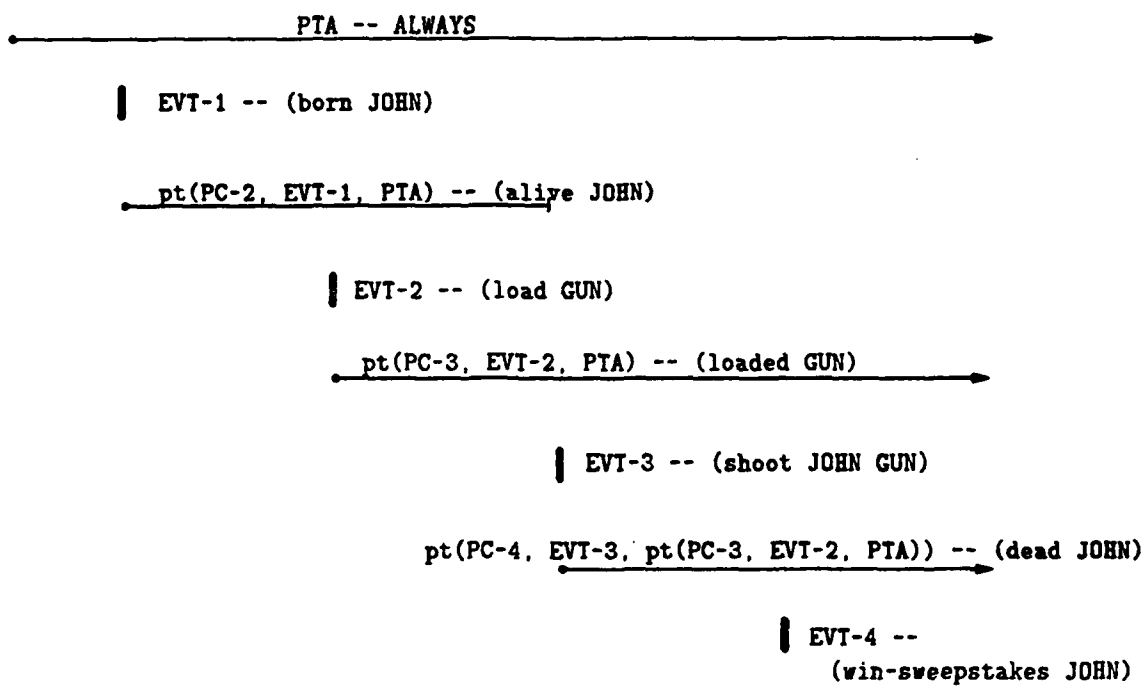


Figure 3: One interpretation of the GUN example

We have four events: ***JOHN*** (a constant symbol representing a particular person) is born, ***GUN*** (a constant symbol representing a particular gun) is loaded, ***JOHN*** is shot with ***GUN***, and ***JOHN*** wins the sweepstakes. Furthermore, we know that the times at which these events happen do not overlap, and that they happen in a particular order. Figure 3 shows a timeline diagram representing one set of conclusions (clippings, point orderings) that might be drawn on the basis of these assertions. We will later show that the state of affairs represented by this line actually models the axioms.

(We're ignoring lots of subtle, and not-so-subtle, points here. For one thing, "shoot" is a

misnomer. "Shoot X with G" should probably read "point gun G at X at very close range and pull the trigger". We furthermore don't bother updating the number of bullets in the gun, *etc.*)

Just to tie together a lot of terminology from several sections back, the union of the temporal axioms $\mathbf{T}$ with a set of problem axioms $\mathbf{P}$ forms the set of wffs $\mathbf{W}$ that, along with the default rule, defines a (Reiter) default theory. Similarly, $(\mathbf{T} \cup \mathbf{P})$ with the modal default rule gives us the desired axiom set for NML, and $(\mathbf{T} \cup \mathbf{P})$ gives us the axiom set $\mathbf{A}$ over which we circumscribe *clipped*. Now we are set to proceed in earnest with the analysis: what conclusions (new persistences, clippings, point orderings) should we draw from these axioms, and are they actually the conclusions licensed by one or more of these default logics?

4 Programming the Axioms

Now we have a set of temporal axioms couched in a default logic, where the logic will license some conclusions and disallow others. Another way to look at the situation is this: the temporal axioms define a particular (infinite) set of models and each logic admits a certain subset of these. As we wrote the axioms we presumably had a model in mind, or at least some characteristics that all "acceptable" models should exhibit. The question is how to express our intent in writing the axioms (our intuitions as to what the "right" deductions are), and whether the intended models are those admitted by one or more of the default logics.

One way to make clear out intentions as to what deductions should follow from temporal and problem axioms is to write a program that performs the deductions. The program would take as input a representation of the problem axioms—a set of *pcause*, *contradict*, and *event* assertions—and would output relations (corresponding to all the temporal predicates) `pcause`, `event`, `persist`, `point`, `contradict`, `clipped`, $\sim$, $\prec$, $\preceq$. To the extent we could then precisely characterize the program's I/O behavior we would also have a precise characterization of the intended models. (We would first have to prove, of course, that the program's output did indeed produce a model of the temporal axioms.)

4.1 Restrictions for simplicity

We're still not at the point, though, where the behavior of such a program is straightforward. Some loose ends need to be tied up—we need to take a stand on how our program will handle certain special cases. Two situations of immediate interest are (1) what to do when the

axioms generate an infinite chain of tokens, and (2) what to do when a particular fact clips its own support.

The first case is easy: consider what happens when we have a causality axiom like

(pcause P E P)

and a *P* fact happens to be true when an *E* event occurs. Conceptually there are an infinite number of tokens generated in such a situation, having the form

pt(pc-tok, evt-tok, ofact-tok),

pt(pc-tok, evt-tok, pt(pc-tok, evt-tok, ofact-tok)),

pt(pc-tok, evt-tok, pt(pc-tok, evt-tok, pt(pc-tok, evt-tok, ofact-tok)))

etc.,

all having the same patterns and coincident begin points.

In writing a program we have to be careful of loops, and detecting loops such as these may be difficult. For example, if we allow events to occur simultaneously, we might have something like this:

(pcause ALWAYS E_1 P)
(pcause P E_2 Q)
(pcause Q E_1 P)

which would cause a loop if an E_1 event occured at the same time as an E_2 event. The chain of *pcause* assertions leading to such a circularity could be arbitrarily long, and since we don't know ahead of time what events might occur simultaneously, it's hard to predict these dangerous situations.

More troublesome, however, is the second problem: a token clipping its own support. A simple example involves the following axioms:

(pcause P E (NOT P))
(contradict (NOT P) P)

Say there's a *P* fact true when an *E* event occurs. As a result, a *(NOT P)* fact is caused, which clips the *P* fact. But at that point there's no justification for the *(NOT P)* fact any more. On the other hand, if the *(NOT P)* fact goes away, then the *P* fact is no longer clipped and the *(NOT P)* fact is licensed to exist again. So either there are no models of the axioms (in particular axiom 1 isn't true), or there is some interval over which contradictory facts hold. As in the first case it may be difficult to detect this situation, since the chain leading from *P* to *(NOT P)* may be arbitrarily long, and may involve an arbitrary number of (simultaneously occuring) events.

Although these are important problems for a practical program to reasons about time (*e.g.* Dean [3]), for us they are technicalities, and obscure the relationship between axioms and program. So we will deal with them by prohibiting such situations altogether. The way we prohibit them is by placing some restrictions on the form of the assertions that can appear as problem axioms. Three such rules do the trick:

1. all event points must be ordered, and must not occur simultaneously. That is, if for some set of problem axioms **P** we have

   $\mathbf{P} \vdash$ *(event tok*$_1$ e_1 p_1*)* and

   $\mathbf{P} \vdash$ *(event tok*$_2$ e_2 p_2*)*

   it must be the case that either

$\mathbf{P} \vdash (\prec p_1\ p_2)$ or

$\mathbf{P} \vdash (\prec p_2\ p_1)$.

2. *Pcause* assertions must be non-circular. That is, for all *pcause* assertions mentioning a particular event type *E*, *(pcause ofact-pat E nfact-pat)*, if *P* appears as *ofact-pat*, then *P* cannot occur as *nfact-pat* for that event type.

3. *Pcause* assertions must not be self-clipping. For all *pcause* assertions mentioning a particular event type *E*, *(pcause ofact-pat E nfact-pat)*, if *P* occurs as *nfact-pat* then no pattern contradictory to *P* can occur as *ofact-pat* in any *pcause* assertion mentioning *E*.

So by forcing events to occur in a well-defined order we eliminate the possibility that cycles and self-clippings happen across more than one event type. Then rules 2 and 3 explicitly prohibit cycles and self-clipping *within* a particular event type.

We'll call any set of problem axioms that satisfies these rules a "well-formed problem-axiom set." Hereafter we'll assume this property of any set of problem axioms.

4.2 Writing the program

Writing the program is pretty straightforward now. The main thing that makes things easier is the restriction that event points be totally ordered. The program can look at the events as being "sorted"—it can process them one at a time, from earliest to latest. Since in our intended model of time events can't affect that part of the world that happened before they occured, we can assume that if a fact hasn't been clipped by any point occuring prior to the current event point, it will *never* be clipped by that point.

The algorithm for processing a single event (assuming all previous events have already been processed) is simple, essentially an exercise in deductive retrieval. Given the event pattern, we look up all pcause assertions that mention it—these will be assertions of the form (pcause ofact-pat evt-pat nfact-pat). For every such pcause we look to see if there are any facts that match ofact-pat—that is, assertions of the form (persist tok ofact-pat bp ep) for any tok, bp and ep. If we find one or more, we next check whether or not they have previously been clipped. For all that have not been clipped, we can generate a new persistence with a pattern nfact-pat.

When we create a new persistence we have to do two more things: perform clippings as needed, and create new persistences as licensed. First we see if there are any contradictory persistences that should be clipped by this new one—we fetch assertions of the form (persist tok potential-clip-pat bp ep) where there is also an assertion of the form (contradict nfact-pat potential-clip-pat). If we find one, and it hasn't already been clipped, we clip it.

The second step in persistence creation—seeing if there are yet more facts that the new fact, combined with current event, can cause—is accomplished by exactly the same machinery as above. If the input obeys the non-circularity restriction, the algorithm terminates in finite time (as there are finite pcauses), though we make no effort to check the input nor to try to detect looping.

The actual program was written in NISP (McDermott [18]), and makes calls to the DUCK deductive retrieval system (McDermott [17]). Appendix B is the code listing and a detailed explanation of the program. Figure 4 contains a timeline representation of the program's output for input representing the sample problem axioms of figure 2 (you'll notice that it is

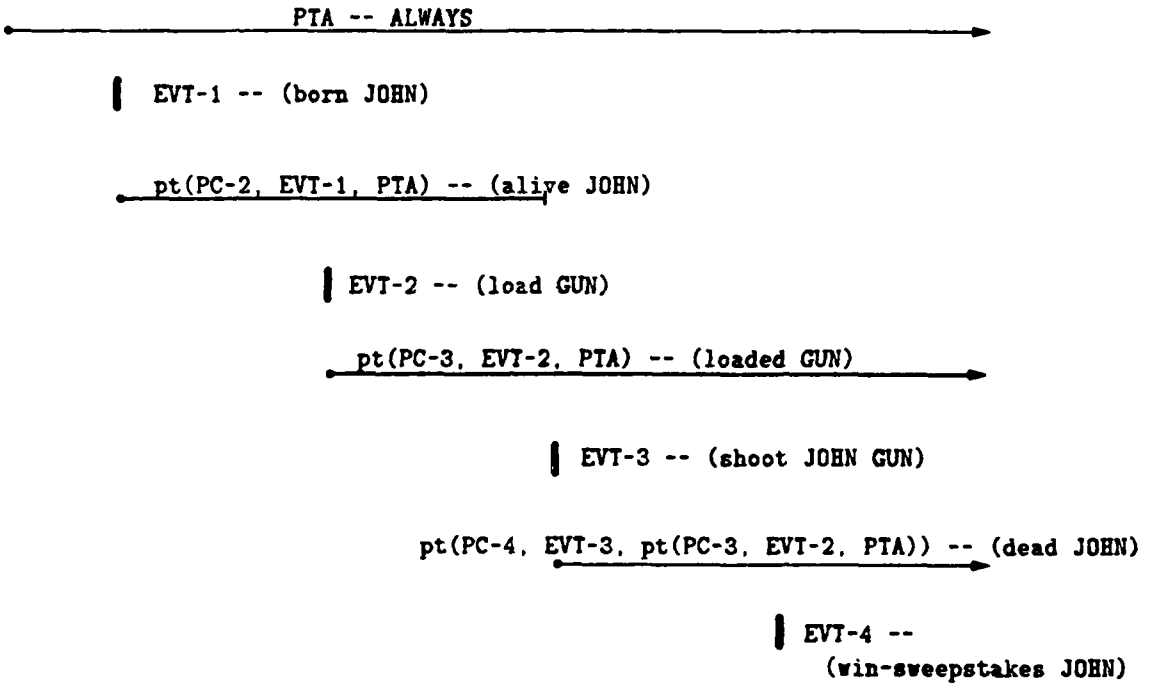


Figure 4: Picture of output for sample problem axioms

the same picture as figure 3 on page 31), and figure 5 is a portion of the interpretation this diagram portrays. Appendix C lists the *actual* program input and output for the "gun" example.

```
persist = { (PTA,
                ALWAYS
                PBA,
                PEA),
             (pt(PC-2, EVT-1, PTA),
                (alive JOHN),
                pb(PC-2, EVT-1, PTA),
                pe(PC-2, EVT-1, PTA)),
             (pt(PC-3, EVT-2, PTA),
                (loaded  GUN),
                pb(PC-3, EVT-2, PTA),
                pe(PC-2, EVT-1, PTA)),
             (pt(PC-4, EVT-3, pt(PC-3, EVT-2, PTA)),
                (dead JOHN),
                pb(PC-4, EVT-3, pt(PC-3, EVT-2, PTA)),
                pe(PC-4, EVT-3, pt(PC-3, EVT-2, PTA)) }

point = { PEVT-1,  PEVT-2,  PEVT-3,  PEVT-4,
          PBA,     PEA,
          pb(PC-2, EVT-1, PTA),
          pe(PC-2, EVT-1, PTA)),
          pb(PC-3, EVT-2, PTA),
          pe(PC-2, EVT-1, PTA)),
          pb(PC-4, EVT-3, pt(PC-3, EVT-2, PTA)),
          pe(PC-4, EVT-3, pt(PC-3, EVT-2, PTA)) }

clip = {  (pt(PC-2, EVT-1, PTA),
              PEVT-3)
          (pt(PC-2, EVT-1, PTA),
             pb(PC-4, EVT-3, pt(PC-3, EVT-2, PTA))),
          (pt(PC-2, EVT-1, PTA),
             PEVT-4),
          (pt(PC-2, EVT-1, PTA),
             PEA),
          (pt(PC-2, EVT-1, PTA),
             pe(PC-2, EVT-1, PTA)),
          (pt(PC-2, EVT-1, PTA),
             pe(PC-4, EVT-3, pt(PC-3, EVT-2, PTA))) }
```

Figure 5: (Part of) interpretation output by program

5 The Program and the Default Logics

Recall that our goal in writing the program was to formalize the deductions that should be licensed by the temporal axioms—to identify which models should be admitted. The program generates an interpretation of the axioms, the relevant part of which was listed in figure 5, but we will have to prove that this interpretation is indeed a model of the axioms.

The immediate problem in trying to compare the two descriptions (the axioms and the program's behavior) is that the two are organized differently. The axioms (the temporal axioms, anyway) are mostly implications with one predicate asserted as a conclusion. So they say "if P and Q and R are true, then S is true." We can thus characterize the "meaning" of a predicate (say S) in terms of the axioms that allow us to conclude that S holds of an individual, and in terms of which *other* axioms mention S in their antecedents.

The program is not organized that way at all. There's no one area of the code, for example, that in isolation represents what happens when a "clipping" occurs. This problem is even more acute in the case of point orderings, since they aren't handled by the deductive retrieval mechanism at all. As a result, it's sometimes hard to compare what the program *does* with what the axioms *say*.

To solve this problem we will develop an intermediate characterization of the program. Our notation in describing the program should be close enough to the code itself so that we can argue convincingly that the description does indeed embody the operations performed by the program. Yet it must also be organized along lines similar enough to the axioms so we can prove some similarity there as well.

5.1 An inductive description of the algorithm

We propose here an inductive description of the algorithm—inductive in that it characterizes the algorithm as a series of stages, one stage for each event in its input. The i^{th} stage represents the state of the program after i events have been processed; the 0^{th} stage is the initial state of the program, and the n^{th} stage (for a problem with n events) describes the program's output.

The inductive description is organized by *relation* (*i.e.* it is a series of descriptions of what individuals are included in particular relations at each stage of the program's computation), and as such we will better be able to establish a correspondence between it and the axioms themselves. But first we must demonstrate that the description indeed describes the program.

To give a sense of what the inductive description looks like, here's the section that describes the `persist` relation (from appendix D, page 99):

1. $\texttt{persist}_0 = \{(\texttt{PTA}, \texttt{ALWAYS}, \texttt{PBA}, \texttt{PEA})\}$
2. $\texttt{persist}_i \subseteq \texttt{persist}_{i+1}$
3. if (pc-tok, pc-ofact-pat, pc-evt-pat, pc-nfact-pat) $\in$ PC
   and there's a token *ofact-tok* with pattern *ofact-pat*,
   and a substitution σ such that $(\texttt{pc-evt-pat})\sigma = \texttt{evt-pat}_{i+1}$,
   and $(\texttt{pc-ofact-pat})\sigma = \texttt{ofact-pat}$,
   and ofact-tok $\in \texttt{unclipped-toks}_i$
   then (pt(pc-tok, $\texttt{evt-tok}_{i+1}$, ofact-tok),
   $(\texttt{nfact-pat})\sigma$,
   pb(pc-tok, $\texttt{evt-tok}_{i+1}$, ofact-tok),
   pe(pc-tok, $\texttt{evt-tok}_{i+1}$, ofact-tok)) $\in \texttt{persist}_{i+1 \setminus i}$.
4. no others

(The notation $x \in \mathcal{R}_{i+1 \setminus i}$ is short for $x \in \mathcal{R}_{i+1} \setminus \mathcal{R}_i$, and can be taken to mean that individual

x is added to relation R at the i+1st stage.)

This part of the description tells us exactly what conditions must hold at the ith stage of the algorithm in order for a particular tuple of individuals to be included in the **persist** relation at the i+1st stage. More particularly, item 1 says that the relation initially contains only the ALWAYS token, and item 2 says that a **persist** assertion is never retracted. Item 3 describes the conditions under which a **persist** assertion may be added at the i+1st stage—*i.e.*, when the i+1st event should cause a new fact. (Note the similarity to axiom 1.) The relation **unclipped-toks**$_i$ is the set of all facts that have not been clipped by any point as of the ith stage. The set **PC** is the **pcause** part of the program's input, and corresponds to the *pcause* assertions in a problem axiom set.

The rest of the inductive description consists of similar sections for all the temporal predicates. In appendix D you'll find the complete description, as well as a "proof" that the description describes exactly the program's output. The (informal) proof is straightforward—for each relation/predicate it's just a matter of examining the program code and verifying that the conditions under which the inductive description of a relation admits a particular tuple of individuals at a particular stage are exactly the conditions under which the program stores the same tuple in the same relation at the same stage.

5.2 The algorithm and the temporal axioms

Now that we've established the correspondence between the program and its inductive description, we needn't bother with the code itself any more. When I refer to "program output" I will mean the output as described inductively.

Next we have to establish a relationship between the description and the axioms themselves. Recall that the program provides an interpretation of the axioms—a universe of individuals and a set of relations corresponding to the temporal predicates.

We have to show that this interpretation is a *model* of the axioms. Formally, what we have to show is this: let **T** be the temporal axioms, and **P** be a well-formed set of problem axioms. For any sentence φ, if $(\mathbf{T} \cup \mathbf{P}) \vdash \varphi$, then φ is true in the interpretation produced by the program when applied to input **P**.

From section D we see that this property is true of all the axioms in **P** by construction, since we used the problem axioms directly as input to the program and the program output them unchanged. Since the axioms in **T** are all implications, what we have to do is go through the axioms in **T** one by one, and show that if the antecedent of the axiom is true in the interpretation, then the conclusion of the axiom is also true in the interpretation.

Let the event assertions in **P** be of the form

(event EVT$_i$ EVT-PAT$_i$ PEVT$_i$) $\qquad i = 1 \ldots n.$

ordered such that $\mathbf{P} \vdash (\prec PEVT_i\ PEVT_j)$ for $i < j$, and let $\mathbf{P}_i$ stand for the set of all *pcause* and *contradict* assertions in **P**, along with the first i *event* assertions. Assuming there are n event assertions, we have $\mathbf{P} = \mathbf{P}_n$.

We then prove by induction on i that the program is a model of $(\mathbf{T} \cup \mathbf{P})$. To establish the base case we can easily show (by reference to the first item in each predicate's inductive description) that the interpretation defined by ($\mathtt{persist}_0$, $\mathtt{clipped}_0$, ...) is a model of $(\mathbf{T} \cup \mathbf{P}_0)$. (This is carried out in detail in appendix E.) Then we assume that at the i^{th} stage the program produces a model of $(\mathbf{T} \cup \mathbf{P}_i)$. Consider axiom 5, for example:

(if (and (persist ?tok ?pat ?bp ?ep)
(clipped ?tok ?p_1))
(≺ ?ep ?p_1))

which we assume is true at the i^{th} stage. We assume that

if (tok, pat, bp, ep) $\in$ $\texttt{persist}_i$, and (tok, p_1) $\in$ $\texttt{clipped}_i$,

then (ep, p_1) $\in$ $\prec_i$.

We make this assumption for *all* the axioms in **T**, then go on to show (again, for all axioms) that the axiom is true at the $i+1^{st}$ stage—*for axiom 5 we prove*

if (tok, pat, bp, ep) $\in$ $\texttt{persist}_{i+1}$, and (tok, p_1) $\in$ $\texttt{clipped}_{i+1}$,

then (ep, p_1) $\in$ $\prec_{i+1}$.

The proof really amounts to an analysis of cases based on the stage in which the tuple of individuals is admitted to the relation; for this example the cases are:

1. (tok, pat, bp, ep) $\in$ $\texttt{persist}_i$ and
   (tok, p_1) $\in$ $\texttt{clipped}_i$
2. (tok, pat, bp, ep) $\in$ $\texttt{persist}_i$ and
   (tok, p_1) $\in$ $\texttt{clipped}_{i+1\backslash i}$
3. (tok, pat, bp, ep) $\in$ $\texttt{persist}_{i+1\backslash i}$ and
   (tok, p_1) $\in$ $\texttt{clipped}_i$
4. (tok, pat, bp, ep) $\in$ $\texttt{persist}_{i+1\backslash i}$ and
   (tok, p_1) $\in$ $\texttt{clipped}_{i+1\backslash i}$

(The notation (tok, pat, bp, ep) $\in$ $\texttt{persist}_{i+1\backslash i}$ means that the tuple (tok, pat, bp, ep) was added to the persist relation at the $i+1^{st}$ stage.)

Now we have to prove each case true, or show why it is impossible. The first is covered by the induction hypothesis. In the second case, we have $(\mathbf{ep}, p_1) \in \prec_{i+1 \setminus i}$ directly by item 4 in the definition of the relation $\prec$. Case three is impossible—a tuple can't be in $\mathtt{clipped}_i$ if it's not in $\mathtt{persist}_i$. Case four violates our restriction on the *pcause* assertions in P, which in effect say that if a fact is created at the $i+1^{st}$ stage (*i.e.* by the $i+1^{st}$ event) it cannot be clipped at the $i+1^{st}$ stage. We relegate a more detailed proof of this to appendix E.

That's the essence of the proof. The rest involves a similar analysis for all the axioms in T. We spare the reader that tedium. For those interested, the full proof appears in Appendix E.

5.3 The algorithm's model and the default logics

The result just proved tells us in effect that the program's output is sound with respect to the temporal axioms—it draws no conclusions contradicting the deductions allowed by the axioms. We noted above that the effect of all three default logics was to restrict models of the temporal axioms in one way or another. So now that we have shown that the algorithm produces a model of the axioms, the next question is whether that model is one of those admitted by the axioms augmented by one or another default rules. For the three systems in question we must consider whether

1. the program's output represents an NML fixed point
2. the program's output represents a DL extension
3. the program's model is minimal in the predicate *clipped*.

Note that we had to be a little careful in our wording above—the model produced by the algorithm is a *semantic* object, while fixed points and extensions are *syntactic*. We will specify in the first two cases exactly how one would take the model and build from it a set of sentences.

5.3.1 Building an NML fixed point

Recall from our discussion of NML in section 2.1 that we're trying to ascertain whether the program's model (properly transformed into an NML theory) is a fixed point of the NM operator—that is, a set **S** such that $\mathbf{S} = NM_{\mathbf{A}}(\mathbf{S})$. The set **A** here is formed from the usual temporal and problem axioms (**T** and **P**), along with the single "default rule"

```
(if (and (persist ?tok ?pat ?bp ?ep)
         (point ?p)
         (M (not (clipped ?tok ?p))))
    (not (clipped ?tok ?p))).
```

Restricting the language L to contain only the predicate symbols mentioned in **T** and **P**, we build an **S** from the program output as follows (recall that the program's output is characterized by relations $\mathtt{R}_n$, where R is a temporal predicate):

1. each sentence φ in $(\mathbf{T} \cup \mathbf{P})$ is in **S**,
2. the default rule above is in **S**,
3. for each relation $\mathtt{R}_n$ output by the algorithm, if $\vec{x} \in \mathtt{R}_n$, then $(R\ \vec{x}) \in \mathbf{S}$,
4. for each of the point-ordering relations P ($\sim$, $\prec$, $\preceq$) and each pair of individuals $\mathtt{p}_1$ and $\mathtt{p}_2 \in \mathtt{point}_n$, if $(\mathtt{p}_1, \mathtt{p}_2) \notin \mathtt{P}$ then *(not (P p_1 p_2))* $\in \mathbf{S}$,
5. for every pair of individuals (tok, p) such that (tok, pat, bp, ep) $\in$ persist for some pat bp and ep, and p $\in \mathtt{point}_n$, if (tok, p) $\notin \mathtt{clipped}_n$, then *(not (clipped tok p))* $\in$**S**.

Now consider applying the NM operator to this set **S**—what we do is add to **A** (which, recall, is the set of temporal and problem axioms plus the default rule) the set of "assumptions" that can be drawn from **S**. The assumptions are contained in a set of formulas of the form *(M q)* where *q*'s negation is not in S. The only interesting such formulas for our purposes (because they're the only ones from which anything else can be deduced) are of the form *(M (not (clipped ...)))*.

From formulas in *M* like the above we might use the default rule to derive formulas of the form *(not (clipped ...))*, except note that by item 5 above any formula of this form that *could* be deduced (because it's negation is not in **S**) is already in **S**. Furthermore, because the algorithm models the temporal axioms (in particular axiom 6 that would allow us to deduce point orderings from formulas of the form *(not (clipped ...))*) any deductions that could be made using formulas in $\mathrm{As}_{\mathbf{A}}(\mathbf{S})$ are already reflected in **S**. Therefore **S** (or more accurately the set $\mathbf{S} \cup \mathrm{As}_{\mathbf{A}}(\mathbf{S})$) is an NML fixed point.

It will be useful later on to note that to establish **S** as a fixed point we used only the information that the relations from which **S** was built were drawn from a model of the temporal axioms. Therefore for any such model $\mathcal{M}$, if one uses the five rules above to build a set of formulas, that set of formulas is an NML fixed point.

5.3.2 Building an extension

A set of sentences (first-order wffs) **E** is an extension of a default theory $\Delta = (\mathbf{D},\mathbf{W})$, according to Reiter's definition [23, p. 89], if it is the smallest fixed point of the operator Γ, defined as follows:

1. $\mathbf{W} \subseteq \Gamma(\mathbf{E})$

2. $\mathrm{Th}(\Gamma(\mathbf{E})) = \Gamma(\mathbf{E})$

3. If $\frac{\alpha : M\,\beta}{\gamma} \in \mathbf{D}$, and $\alpha \in \Gamma(\mathbf{E})$ and (not β) $\notin \Gamma(\mathbf{E})$, then $\gamma \in \Gamma(\mathbf{E})$.

We build the proposed extension **E** in exactly the same way we build the NML fixed point:

1. each sentence φ in $(\mathbf{T} \cup \mathbf{P})$ is in **E**,
2. the default rule above is in **E**,
3. for each relation $\mathtt{R}_n$ output by the algorithm, if $\vec{x} \in \mathtt{R}_n$, then $(R\ \vec{x}) \in \mathbf{E}$,
4. for each of the point-ordering relations P ($\sim$, $\prec$, $\preceq$) and each pair of individuals $\mathtt{p}_1$ and $\mathtt{p}_2 \in \mathtt{point}_n$, if $(\mathtt{p}_1,\mathtt{p}_2) \notin$ P then *(not (P p_1 p_2))* $\in \mathbf{E}$,
5. for every pair of individuals (tok, p) such that (tok, pat, bp, ep) $\in$ persist for some pat bp and ep, and p $\in \mathtt{point}_n$, if (tok, p) $\notin \mathtt{clipped}_n$, then *(not (clipped tok p))* $\in \mathbf{E}$.

It's easy to see that **E** is a fixed point of the Γ operator: first of all, $\mathbf{W} = (\mathbf{T} \cup \mathbf{P}) \subseteq \mathbf{E}$ by construction. Since **E** is a model of $(\mathbf{T} \cup \mathbf{P})$, and since T contains the only implications in **E**, **E** is its own deductive closure. Finally, for any fact token tok and point p, **E** contains by construction either *(clipped tok p)* or *(not (clipped tok p))*, thus satisfying the third requirement.

To see that **E** is the smallest such fixed point, consider $\mathbf{E}' \subset \mathbf{E}$, and a wff α that's in **E** but not in **E'**. Since $\alpha \in \mathbf{E}$, it must have come either from **T**, from $\mathbf{P}_n$, or from the program's output (i.e., it's a sentence of the form $(R\ \vec{x})$, where R is a temporal prediate). But if it's the case, for example, that $\alpha \in \mathbf{T}$ and $\alpha \notin \mathbf{E}'$, then $\mathbf{W} \notin \mathbf{E}'$, and **E'** is not a fixed point of Γ. Taking α from either of the other two sets that comprise **E** results in **E'**

not containing its own deductive closure, or not embodying the default rule, respectively. Thus **E** is the smallest fixed point of Γ, and therefore an extension of the temporal default theory.

5.3.3 Establishing a model minimal in clipped

Let $\mathcal{M}$ be the model produced by the program (see figures 3 and 5). To show that $\mathcal{M}$ is minimal in **clipped** we must show that there's no model $\mathcal{M}'$ whose extension in **clipped** (call it $\texttt{clipped}_{\mathcal{M}'}$) is a proper subset of the extension of **clipped** in $\mathcal{M}$ (call it $\texttt{clipped}_{\mathcal{M}}$).

So assume that $\mathcal{M}'$ is such a submodel. Since we're allowing the other temporal predicates to vary as parameters, the temporal relations in $\mathcal{M}'$ needn't contain the same individuals as those in $\mathcal{M}$, but since $\mathcal{M}'$ is assumed to be a model, those relations must satisfy the temporal and problem axioms.

First of all, let's recall (using some notational shorthand) the clippings in $\mathcal{M}$:

ALIVE fact clipped by these points:
the **SHOT** event point
the **DEAD** fact begin point
the **WIN** event
the **ALWAYS** fact end point
the **LOADED** fact end point
the **DEAD** fact end point.

If $\mathcal{M}'$ is to satisfy the temporal axioms, several things must be true (as they are true in $\mathcal{M}$): first of all it must contain the **ALWAYS** fact and its endpoints, it must contain all the event points and their orderings as they appear in **P**. From this information we can use axiom 1 to conclude that the $\texttt{persist}_{\mathcal{M}'}$ must also contain the **ALIVE** and **LOADED** facts. Further, note that the **LOADED** fact cannot be clipped by the **SHOT** event point, because if it

were $\mathtt{clipped}_{\mathcal{M}'}$ would not be a proper subset of $\mathtt{clipped}_{\mathcal{M}}$. But if that's the case, axiom 6 says that the SHOT event occurs before the end of the LOADED fact, and hence by axiom 1 the DEAD persistence has to exist in $\mathcal{M}'$. But if $\mathtt{persist}_{\mathit{clipped}_{\mathcal{M}'}}$ contains the DEAD persistence, and since $\mathcal{M}'$ has to embody all the *contradict* axioms in $\mathbf{P}$ (including the one that says that being dead clips being alive), it must be the case that the begin point of the DEAD fact clips the ALIVE fact in $\mathcal{M}'$.

But once we have this "initial" clipping, all the other clippings in $\mathcal{M}$ follow directly from axiom 4. So either $\mathcal{M}'$ is not a model of $(\mathbf{T} \cup \mathbf{P})$, or it's not a submodel of $\mathcal{M}$ in **clipped**. Therefore $\mathcal{M}$ is minimal in **clipped**.

6 Uniqueness of the Algorithm's Model

We have noted that the program produces an interpretation for a set of temporal and problem axioms, that the interpretation is a model of those axioms, and that the model can be used to build an NML fixed point and a DL extension, and that the model is minimal in the predicate *clipped*. But recall our discussion of section 2, in which noted that the existence of such minimal models (here I'm using the term generically to mean either a fixed point, extension, or minimal model depending on the logic of choice) is of little use unless that model is the *only* one. So the question to be considered now is whether this interpretation is the *only* model, represents a *unique* NML fixed point or DL extension, and is the *only* model minimal in *clipped*.

The answer to all these questions is *no*, as we shall show by presenting a counterexample—a second interpretation of the theory consisting of the temporal axioms along with the problem axioms for the gun example (figure 2) that is also a model minimal in *clipped*, and can be used to construct a different fixed point and extension by the same methods we used in the previous section.

Consider the state of affairs pictured in figure 6. The interesting difference between this situation and that of figure 4 is that the *loaded* persistence ends sometime before the shot is fired, thus preventing the *dead* persistence from being created, thus preventing the *alive* persistence from being clipped.

To look at it from the "point of view" of the *alive* persistence, we might say that the default rule concerning clipping indicates that the *alive* persistence wants to remain unclipped if possible. But if the *dead* persistence comes into being it is compelled by axiom 3 to clip

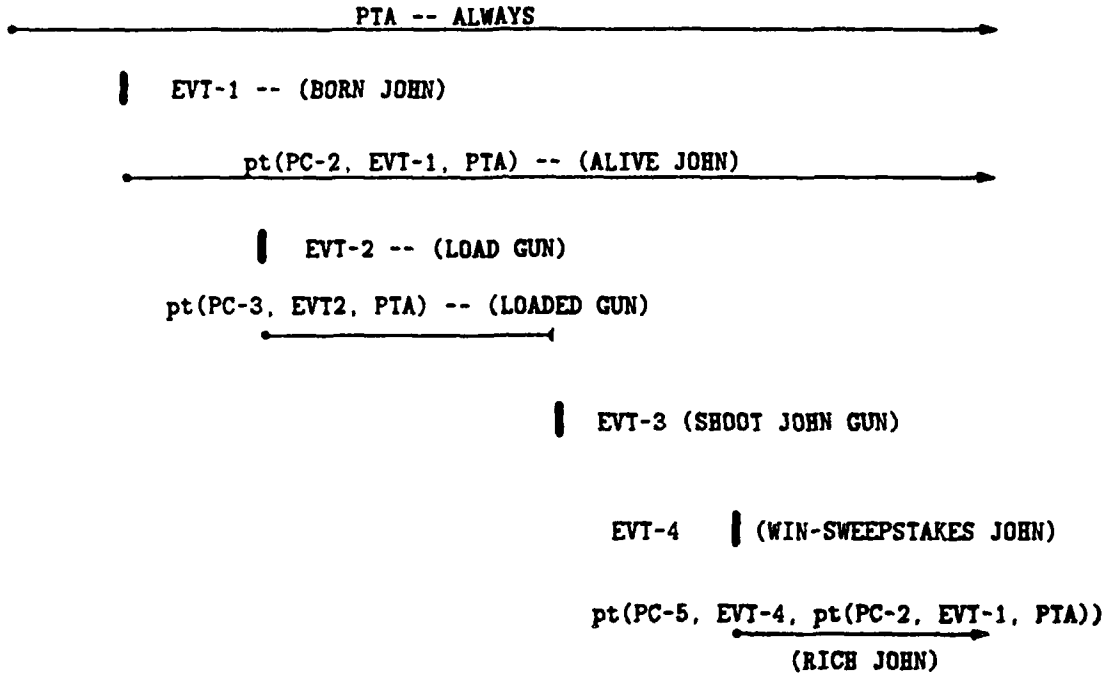


Figure 6: Alternative model of sample problem

the *alive* persistence. Well the *dead* persistence must be caused by the *SHOOT* event if it's the case that the *loaded* persistence endures past the time that event occurs. But if the *loaded* persistence does *not* endure past the time of the shot, it must be clipped by the time point representing the instant at which the shot happened. And so we arrive at the state of affairs pictured in figure 6.

A simpler example might make clearer the sort of situation we're describing here: if one were asked "if I light a match then touch it to a candle wick, will the candle light?" One (arguably) reasonable answer might be "either the match goes out *before* you touch it to the wick, or the candle will light." The situation of figure 4 is analogous to the disjunct in which the candle lights; figure 6 is like the disjunct in which the match goes out.

So what do we make of this state of affairs? Is it a model, and, if so, is it minimal in *clipped*? Can it be used, in the same way we used the program's model, to build a fixed point or extension? We hope not: the picture represents a state of affairs in which *something* happened to clip the *loaded* persistence, but there's nothing (no event, no fact, no *contradict* information) to justify that clipping. We would rather not admit into our theory of temporal reasoning the possibility of things happening without explicit cause.

Unfortunately, this situation *is* a model, the model *is* minimal in *clipped*, and we *can* use the procedures of the previous section to build a fixed point and an extension.

6.1 Establishing another model

To show that we have a model we must show the following sorts of things:

1. any time there are appropriate facts, events, and causality, a new fact is caused (*i.e.* axiom 1 is satisfied)

2. any time there are appropriate facts and contradiction axioms, a fact is clipped (axiom 3)
3. once a fact is clipped it "stays clipped" (axiom 4)
4. if a fact is not clipped its endpoint endures (axiom 6)
5. the point-ordering relations (transitivity, symmetry, substitution) are satisfied.

We'll skip a detailed demonstration that all the axioms are indeed satisfied. One can convince oneself of this pretty much by looking at the picture in figure 6: the *ALWAYS* fact indeed causes the *ALIVE* and *LOADED* facts; and given that the *LOADED* fact is clipped by the *SHOOT* event, it is indeed clipped by all subsequent points. The endpoints of the unclipped persistences endure beyond all other points. And so on.

6.2 Establishing another fixed point

We can build a set of sentences in the manner we did for the algorithm's model: include the sentences in **T** and **P**, add sentences corresponding to each member of each temporal relation, and for each `tok` and `p` such that `tok` participates in a tuple in `persist` and `p` $\in$ `point` and `(tok, p)` $\notin$ `clipped`, add the sentence *(not (clipped tok p))*. We showed in section 5.3.1 that the resulting set of sentences is a fixed point.

6.3 Establishing another minimal model

We will use the procedure of section 5.3.3 to establish minimality. Let $\mathcal{M}$ be the alternative model of (**T** $\cup$ **P**)(the one pictured in section 6). Its extension of `clipped` contains the following pairs:

`LOADED` fact clipped by these points:

the SHOT event point
the WIN event
the RICH fact begin point
the ALWAYS fact end point
the ALIVE fact end point
the RICH fact end point.

Again let $\mathcal{M}'$ be a submodel of $\mathcal{M}$ in clipped. $\mathcal{M}'$ must contain the ALWAYS and ALIVE facts too. It must also be the case that in $\mathcal{M}'$ the SHOT event clips the LOADED fact, because otherwise the DEAD fact would come into existence and clip the ALIVE fact, and $\texttt{clipped}_{\mathcal{M}'}$ would not be a proper subset of $\texttt{clipped}_{\mathcal{M}}$. But once again, once we establish this initial clipping, all the other clippings follow from axiom 4. Thus there can be no submodel of $\mathcal{M}$ in clipped, and $\mathcal{M}$ is minimal.

6.4 Establishing another extension

We build a potential extension **E** as we did a potential fixed point, including sentences in **T** and **P**, sentences corresponding to tuples in the temporal relations, and *(not (clipped ...))* sentences as appropriate.

The process of verifying that **E** is indeed an extension is a complicated one, but it also turns out to shed light on the reason we find multiple extensions to these theories, so we will develop the argument in some detail.

First we should establish some notational shortcuts in referring to the various persistences and points in the theory. Using names such as *pe(PC-5. EVT-4. pt(PC-2. EVT-1. PTA))* is as annoying for the author to type as it is for the reader to recall what it stands for. Our intent is that the reader should be able to understand the proof from looking at the

axioms and at the picture (figure 6), so we will introduce some more descriptive names for the persistences and points. The persistence named by *pt(PC-2. EVT-1. PTA)* has the pattern *(alive JOHN)*, for example. Since the pattern is easy to remember, and since *JOHN* is irrelevant, we will refer to this token as *tok(ALIVE)* and to its begin and end points as *begin(ALIVE)* and *end(ALIVE)*, respectively. As a result, the theory contains the following persistences: *tok(ALWAYS)*, *tok(ALIVE)*, *tok(LOADED)*, and *tok(RICH)*, and corresponding points *begin(ALWAYS)*, *end(ALWAYS)*, *etc*. Points at which events happen will be *pt(BORN)*, *pt(LOAD)*, *etc*.

To verify that $\mathbf{E}$ is an extension we'll use Reiter's theorem 2.1 [23, p. 89], which says that $\mathbf{E}$ is an extension if it is equal in the limit to the series $\mathbf{E}_0$, $\mathbf{E}_1$, ..., where $\mathbf{E}_0$ in our case is $(\mathbf{T} \cup \mathbf{P})$, and you get from $\mathbf{E}_i$ to $\mathbf{E}_{i+1}$ by the following steps:

1. Find all the pairs *(tok, p)* such that *tok* is a persistence in $\mathbf{E}_i$ and *p* is a point in $\mathbf{E}_i$, and *(clipped tok p)* is not in $\mathbf{E}$. (In short, this step identifies all potential applications of the default rule.)
2. Form the set $\mathbf{E}_i \cup \{$*(not (clipped tok p))* $\}$ for all the *tok* and *p* above.
3. Take the deductive closure.

Note that $(\mathbf{T} \cup \mathbf{P}) \subseteq \mathbf{E}_0 \subseteq \mathbf{E}_1 \subseteq \ldots \subseteq \mathbf{E}$. Since the first step in forming $\mathbf{E}_{i+1}$ makes use of the clippings in the *eventual* extension $\mathbf{E}$, the verification process is really one of proposing an extension complete with clippings, starting with the problem axioms, and showing that all these clippings are "justified" (*i.e.* they all follow from the first-order axioms along with all default-rule instances that aren't contradicted by the chosen clippings).

For this example the clippings in question are as follows:

tok(LOADED) is clipped by *pt(SHOOT)*
pt(WIN)

begin(RICH)
end(ALWAYS)
end(ALIVE)
end(RICH)

But recall from our discussion of minimal models that the first clipping is really the critial one—if we can show that *tok(LOADED)* is clipped by *pt(SHOOT)*, the rest of the clippings follow from axiom 4, and we are done.

So we start with the set $\mathbf{E}_0$, which includes the single persistence *tok(ALWAYS)* and the points *pt(BORN)*, *pt(LOAD)*, *pt(SHOOT)*, *pt(WIN)*, *begin(ALWAYS)*, and *end(ALWAYS)*. To get $\mathbf{E}_1$ we take the deductive closure of $\mathbf{E}_0$ along with all the "non-clippings" that involve *tok(ALWAYS)* and the five points above.

From these formulas we can deduce two new persistences, *tok(ALIVE)* and *tok(LOADED)*, along with lots of point orderings. Some of the interesting sentences in $\mathbf{E}_1$ are:

(1) *(persist tok(ALIVE) ...)*
(2) *(persist tok(LOADED) ...)*
(3) *(~ begin(ALIVE) pt(BORN))*
(4) *(~ begin(LOADED) pt(LOAD))*.

Note, however that we *can't* deduce that *tok(DEAD)* is a persistence, because we can't deduce that *(≤ pt(SHOOT) end(LOADED))*.

The interesting default instance that we *do* add in moving from $\mathbf{E}_1$ to $\mathbf{E}_2$ is the sentence

(5) *(not (clipped tok(ALIVE)) pt(SHOOT))*,

from which we have to try to deduce

(*) *(clipped tok(LOADED) pt(SHOOT))*.

What we want to show, then, is that (*) follows from $\mathbf{E}_1 \cup (5)$.

We first consider what the world would have had to be like in order that *tok(DEAD) not* become a persistence. So we look at the contrapositive of axiom 1, substituted for that individual:

```
      (if (or (not (persist tok(DEAD) ...)
              (not (~ begin(DEAD) pt(SHOOT))))
(6)       (or (not (pcause LOADED SHOT DEAD))
(7)           (not (persist tok(LOADED) ...))
(8)           (not (event SHOT))
(9)           (not (≤ begin(LOADED) pt(SHOOT)))
(10)          (not (≤ pt(SHOOT) end(LOADED))))).
```

But note that the negation of sentences (6) and (8) are in $\mathbf{P}$, and the negation of sentences (7) and (9) follow from $\mathbf{E}_1$. (Deduce (9) from (4), from the event-point orderings in $\mathbf{P}$, and from transitivity of $\preceq$.)

So we can reduce the above implication to

```
(11)    (if (or (not (persist tok(DEAD) ...)
(12)            (not (~ begin(DEAD) pt(SHOOT))))
(13)        (not (≤ pt(SHOOT) end(LOADED)))).
```

Now we explore the consequences of (13), writing the contrapositive of axiom 6 with substitution:

```
(13)    (if (not (≤ pt(SHOOT) end(LOADED)))
(14)        (or (not (persist tok(LOADED) ...))
(15)            (not (point pt(SHOOT)))
(*)             (clipped tok(LOADED) pt(SHOOT))))
```

This time the negation of (14) is in $\mathbf{E}_1$, and the negation of (15) follows from the *event* axiom in $\mathbf{P}$ along with axiom 8. So we can combine the two implications above to get

(11) *(if (or (not (persist tok(DEAD) DEAD begin(DEAD) end(DEAD))*
(12) *(not (~ begin(DEAD) pt(SHOOT))))*
(*) *(clipped tok(LOADED) pt(SHOOT)))).*

In other words, if $\mathbf{E} \vdash$ (11), or if $\mathbf{E} \vdash$ (12), then $\mathbf{E} \vdash$ (*).

Now we'll work in the other direction, and see what we can deduce from the interesting default instance:

(5) *(not (clipped tok(ALIVE) pt(SHOOT))).*

Substituting into the contrapositive of axiom 4 we get

(5) *(if (not (clipped tok(ALIVE) pt(SHOOT)))*
(16) *(or (not (persist tok(ALIVE) ALIVE begin(ALIVE) end(ALIVE)))*
(17) *(not (clipped tok(ALIVE) ?p1))*
(18) *(not (point ?p1))*
(19) *(not (point pt(SHOOT)))*
(20) *(and (not (≺ ?p1 pt(SHOOT)))*
(21) *(not (~ ?p1 pt(SHOOT))))))*

(Recall that *?p1* indicates a universally quantified variable.) We can eliminate the antecedent (since it's the default we're adding), along with (16) (which is in $\mathbf{E}_1$) and (19) which follows from the *event* axiom and axiom 8, so we're left with

(or (not (clipped tok(ALIVE) ?p1))
(not (point ?p1))
(and (not (≺ ?p1 pt(SHOOT)))
(not (~ ?p1 pt(SHOOT)))))

and substituting *begin(DEAD)* for *?p1* we get

(22) *(or (not (clipped tok(ALIVE) begin(DEAD)))*
(23) *(not (point begin(DEAD)))*
(24) *(and (not (≺ begin(DEAD) pt(SHOOT)))*
(not (~ begin(DEAD) pt(SHOOT))))).

From (23) we can use axiom 8 (contrapositive) to derive (11), and thus derive (*). And note that (12), and thus (*), follows directly from (24). So either $\mathbf{E} \vdash$ (22), or $\mathbf{E} \vdash$ (*).

Exploring the implications of (22), we take the contrapositive of axiom 3:

(22) *(if (not (clipped tok(ALIVE) begin(DEAD)))*
(25) *(or (not (persist tok(ALIVE) ...)*
(11) *(not (persist tok(DEAD) ...)*
(27) *(not (contradict ALIVE DEAD)*
(28) *(not (≺ begin(ALIVE) begin(DEAD)))*

In this case the negation of (25) is in $\mathbf{E}_1$, (11) implies (*), and the negation of (27) is in $\mathbf{P}$. So it must be the case either that $\mathbf{E} \vdash$ (*) or that $\mathbf{E} \vdash$ (28).

Plugging (28) into the contrapositive of axiom 15b (along with a substitution) yields:

(28) *(if (not (≺ begin(ALIVE) begin(DEAD)))*
(29) *(or (not (∼ pt(SHOOT) begin(DEAD)))*
(30) *(not (≺ begin(ALIVE) pt(SHOOT)))))*

where (29) implies (12) by the symmetry of $\sim$.

But recall that we have in $\mathbf{E}_1$ the sentence

(3) *(∼ begin(ALIVE) pt(BORN)).*

and that in $\mathbf{P}$ we have axioms ordering the event points, in particular

(≺ pt(BORN) pt(SHOOT)).

So we have by the substition axiom for $\sim$ that $\mathbf{E}_1$ entails the negation of (30):

(≺ begin(ALIVE) pt(SHOOT)).

So it must be the case either that $\mathbf{E}_1 \vdash$ (*) or that $\mathbf{E}_1$ is inconsistent. And from (*), as we noted, it follows that the situation of figure 6 is an extension of $(\mathbf{T} \cup \mathbf{P})$.

7 Analysis

In our intial discussion of the three nonmonotonic logics, before describing our temporal representation, we expressed the hope that our temporal theory would be exempt from certain difficulties, notably the admission of multiple states of the world (multiple fixed points, extensions, minimal models).

This turned out not to be the case—in fact the situation we find ourselves in is even worse than the problem of multiple fixed points we described in section 2. We saw multiple extensions in the Nixon example, but they both seemed somehow plausible: in one case his "Quakerhood" won out over his "Republicanhood", thus we concluded he was a pacifist, and in the other case his Republican nature prevailed. Practical difficulties aside (meaning that we have no way to choose between extensions or explore one of them in isolation), we have to admit that *either* extension represents a reasonable way of looking at the world, given the premises.

Our temporal theory admits no such ambivalence. We were clear about the conclusions that should be drawn from the axioms, as evidenced by the precise way our program selected a model. For a given set of problem axioms there is only one valid world state.

The result of section 6 was unexpected. We had noted in section 2 that the form of the default rules necessary to represent the enduring nature of persistences didn't seem to be the sort of default rules that led to multiple extensions. Recall that in the Nixon example we had one default rule that "wanted" to assert *pacifist* of an individual, and another rule that "wanted to" assert *non-pacifist*. In the temporal domain there was only one rule, so, we hoped, there would be no conflicting default inferences to be drawn about any individual.

The questions then are two: what is it about this nonmonotonic theory that leads to multiple extensions? The answer to that question prompts another: is this quality inherent in the temporal domain, or might we avoid it by carefully restating the theory? If it *is* essential to a temporal logic, is it essential to other represention problems as well?

7.1 Interacting default instances

Let's return to the "gun" example, and the program's model (figure 4, page 38). Consider how we were able to deduce the clipping

(clipped tok(ALIVE) begin(DEAD)).

Aside from the obvious *contradict* assertion, the main thing we had to have (to satisfy the antecedent of axiom 3) was a formula of the form *(persist tok(DEAD) ...)*. And how were we able to deduce such a formula? In addition to appropriate formulas involving *event* and *pcause*, the antecedent of axiom 1 required that the causing event (the shot) happen during the span of a *LOADED* persistence. That is, it had to be the case that *($\preceq$ pt(SHOOT) end(LOADED))*. But to deduce that point ordering (through axiom 6) we had to jump to the conclusion

(not (clipped tok(LOADED) PEVT-3)).

In short, in order to deduce a clipping we had to assume a non-clipping.

And here we see the negative interaction we hoped to avoid. It arises not as two default rules that advise contradictory conclusions, but in the relationship holding between two

individuals *within* the theory, such that invoking an instance of the default rule for one involves being unable to apply an instance of the default rule for the other.

Looked at from the standpoint of our circumscribed axioms, we might say that our aim in trying to establish a minimal model is to "save as many clippings as possible." In other words, if we have a choice between putting a particular pair of individuals into the `clipped` relation for a particular model, we would prefer not to do so. (So in the "gun" example we chose to assume that PEVT-3 did *not* clip the LOADED persistence.) The problem with these negatively related defaults is that this decision may then require us at some "later" point to put another pair of individuals in the relation (as we then had to let the DEAD persistence clip the ALIVE persistence). Conversely (as we did in the second model) we could choose to let the third event clip the LOADED fact, and in doing so we would save an "earlier" clipping. As we noted, both models are minimal; the logic cannot favor one over the other.

7.2 Ordered individuals

In the above discussion we glossed over another important feature of the problem. Twice we referred to a "later point" or an "earlier clipping", but didn't make precise what we meant by "later" and "earlier."

Since we're working with time, the definition is deceptively easy: a later point is one that occurs later in time; a later fact is one that *begins* later in time. But this is a very important concept, since it is exactly this ordering of the individuals, imposed by the point-ordering predicates, that's the key to understanding why we as "temporal modellers" favor one model (extension, fixed point) over another, but why the logics are incapable of expressing that preference.

Consider this example, which very simply points out the features of the domain that lead to multiple extensions: negatively interacting default rules, coupled with ordered individuals.

We'll take a language that has as individuals only the set $\{a_0, a_1, a_2, a_3\}$, and the predicates *s*, and *ab*. The relation *s* will be successor—it will induce a total order on the individuals—and *ab* will be a one-place predicate, and will stand for "abnormality" of some sort (in the style of McCarthy [13]). Our default rule will try to express the concept "assume non-abnormality, lacking evidence to the contrary." So to do default inference we will circumscribe over *ab* (letting *s* vary as a parameter), or we will add the sentence

```
(if (M (not (ab ?x)))
    (not (ab ?x))),
```

or we will use the default rule schema

$$\frac{: \mathbf{M}\ (not\ \ (ab\ ?x))}{(not\ \ (ab\ ?x))}.$$

Here are the rest of the axioms:

```
; define the ordering

(1) (iff (s ?x ?y)
         (or (and (= ?x a1) (= ?y a0))
             (and (= ?x a2) (= ?y a1))
             (and (= ?x a3) (= ?y a2))))

; if an individual is not abnormal, then its successor is abnormal

(2) (if (and (not (ab ?x))
             (s ?y ?x))
        (ab ?y))
```

; individual a_0 is not abonormal

(3) *(not (ab a_0))*

We can easily enumerate all models of this theory. First of all, the extension of *s* is fixed to include exactly the three pairs enumerated in the first axiom, so we will not consider it further. The extension of *ab* must always contain a_1, and must never contain a_0. The possible extensions of *ab*, then, are the following:

(a) $\{a_1, a_3\}$

(b) $\{a_1, a_2\}$

(c) $\{a_1, a_2, a_3\}$

Note that (a) and (b) are both minimal in *ab* (neither is a subset of the other) but that (c) is not. (It's also the case that (a) and (b) are NML fixed points and DL extensions, but we will not prove that.) Model (a) corresponds to the program's model of the gun example, in that it's the result of fixing the *ab* property of the individuals in the order imposed on them by the *s* predicate. That is, we first decide on the status of a_0, then of a_1. We have no choice in either case. Next we decide on a_2, and since we *do* have a choice we choose that it *not* be *ab*. Once we've made that decision, axiom (2) forces us to choose a_3 as *ab*.

We can get to model (b) by working in reverse order. First we decide that a_3 *not* be *ab*. The contrapositive of axiom (2) then requires that a_2 be *ab*, and as noted above we have no choice with regards to a_1 or a_0.

This second example also makes more clear why the second model of the gun axioms

did in fact turn out to be an extension. We had originally hoped that this would not be the case. We recalled the definition of an extension, particularly the fact that every wff in the extension must follow from the first order axioms (the set $(\mathbf{T} \cup \mathbf{P})$ in this case), along with licensed application of the default rule. We saw only two ways of deriving a clipping from axioms in T: directly from the begin point of a contradictory token, and indirectly from another point, once the initial clipping had been derived. We looked at the alternative model, and saw that the initial clipping (the clipping of the ***LOADED*** by the ***SHOOT*** event fit neither of these criteria). So we hoped that we could prove that there was *no* justification for this clipping, and therefore that the model could not represent an extension.

What we failed to recognize initially is made clear in the *ab* example: in model (b) the contrapositive of axiom (2) tells us that once we chose a_3 to be non-*ab* it *had* to be the case that a_2 *was ab*. In fact, the contrapositive of axiom (2) is exactly the justification of the wff $(ab\ a_2)$ that proves that model (2) is indeed an extension. Translating back to the gun example, we can now see the line taken by our proof of "extensionhood" for the alternate model: the (default) assumption that the ***ALIVE*** fact was *not* clipped by the ***DEAD*** fact *itself* allowed us to deduce that the ***ALIVE*** fact must have been clipped by the ***SHOOT*** event.

7.3 Implications for the temporal domain

So now we've pointed out two ways of building a minimal model for the *ab* example (and for the gun example too, for that matter): one starts at one end or the other of the ordering of individuals, and when faced with a choice as to whether an individual should be *ab* or not (*clipped* or not), one chooses negatively (in accordance with the default rule). Whichever

end you begin at, you eventually arrive at a minimal model.

So should we choose one end of the order to start at over another? As we expressed in sections 2 and 3, our goal in writing the temporal axioms was to deduce the effects of certain events happening in time, coupled with statements about causality and contradiction. We might call this problem "temporal projection." And for the temporal projection problem we are bound by our notion of causality to the view that events in the present can affect the course of events in the future, but not *vice versa*. Thus we are bound to accept the "past-to-future" model, but no others.

To see what this means in terms of the logics, let's couch the *ab* example in temporal terms. Consider the successor predicate to indicate fatherhood—*(s x y)* is intended to mean "x is the father of y". Axiom (2) reflects our knowledge about some genetic phenomenon: if an individual doesn't exhibit a particular trait (the "ab" trait), then his son will. Or put more specifically, a father's lack of "ab" *causes* his son to be "ab". The contrapositive of axiom (2), then, could be read to mean that a person's lack of "ab" *caused his father to have had the trait*. Obviously we want to allow the first such inference but not the second.

We've now mentioned two ways to get minimal models out of this sort of theory (start from the beginning and move forward, and start from the end and move backward), but it should be clear that there are other, even less intuitive minimal models as well. In fact, one might start *anywhere* in the order of individuals, fix it as "normal", then simultaneously work forward (inferring what this normality implies) and backward (inferring what must have happened in order for this individual to have been normal). In any case, it's clear that these models will all describe states of the world that are internally consistent but incomparable, and also that only one of the models is an accurate portrayal of our intentions

when we wrote the temporal axioms.

7.4 Set inclusion as a minimality criterion

We had initially thought that the models problem might be due to the particular way we expressed the temporal axioms—that an alternative phrasing might make the problem go away. The "father" example above provides strong evidence to the contrary: conflicting default instances and ordered individuals seem essential to any temporal theory, and these theories seem inevitably prone to multiple interpretations.

We might also wonder whether the default logics might be coerced somehow to accept the intended model as the only valid one. Perhaps circumscribing over *ab* was wrong, and we might define some other predicate (or predicates) that are minimal in the circumscriptive sense just in case the temporal predicates have extensions corresponding to the past-to-future interpretation produced by our program.

The problem seems to be that "minimal in the circumscriptive sense" is inexorably tied to the notion of set inclusion. Recall that a model minimal in a predicate *P* is minimal because no other model's extension of *P* includes fewer individuals [3]. But recalling the models of the "father" example above (p. 7.2), no notion of set inclusion seems to capture the difference between the alternative models. In fact, the notion of minimality expressed by the program (start with the first individual, assume that it's non-abonormal if consistent to do so, make inferences as warranted by that assumption, then go on to the next individual in the order) seems unrelated to any notion of set inclusion. In any event, we have been unable

[3]The notion of "prioritized circumscription" (McCarthy [13], Lifschitz [11]) also seems bound to the notion of minimality measured by set inclusion.

to find any predicate that has minimal extension just in case the program's interepretation is satisfied, and must conclude that there is no straightforward way to express in the logic the notion of minimality inherent in the program's inductive description.

8 Conclusion

We noted at the outset of this paper the paucity of results applying default logics to practical representation problems in AI. Researchers in the field have either avoided talking about the problem altogether, or have gone ahead and used a default logic, with the implicit assumption (or hope) that since it was clear what they *meant* to express in their theory, the semantics of their chosen formal system would eventually be understood and would correspond to their wishes. While the "multiple models" problem has always been recognized as a technical problem with default logics, ([19,15,23]), it has always been hoped that the problem would not manifest itself in practical applications. (It was always recognized that there might be more than one model, but it was hoped that each would at least have some intuitive justification, as in the Nixon example.)

So McDermott in his temporal logic [16, p. 121] uses the M operator, along with an explanation of what idea he *intends* that it capture. McCarthy claims [13, sect. 9] that circumscription solves a particular version of the frame problem. Joshi [10] couches rules for conversational interaction in terms of default rules in Reiter's logic, along with an English-language explanation of *what they are supposed to mean.* In each case the researcher has in his own mind a clear idea of what conclusions his theory licenses—what conclusions *should* be allowed by the logic—and the syntax of the default logics (seem to) provide a parsimonious way for him to express those intentions.

We have shown that for a particular problem, that of temporal projection, the assumption that the semantics of nonmonotonic logics will "work out right" is a bad one. We were able to express our temporal theory in a default logic, and able to explain inductively

the conclusions we intended the inference mechanism to draw. But these were not the conclusions licensed by any of the default logics. This problem seems to be inherent in the temporal representation, and plagues all the default logics we know of. We must conclude, therefore, that McDermott's temporal logic does *not* have its intended meaning, and that circumscription (in its present form) does *not* solve the frame problem.

Other temporal ontologies have avoided couching their theories in a default logic. Hayes, for example (in [9]), introduces the notion of a *history*—a continuous chunk of space/time. The piano in my living room, for example, is represented by a "piano in the living room history" that has both a spatial and temporal extent. Temporal projection must eventually show up in Hayes' theory in deducing the duration of these histories. So if I see the piano in the living room this morning, I may want to project that its history endures past the time that I actually observe it. (Note the similarity to the persistence of facts.) But such a conclusion cannot follow from Hayes' system (as it is expressed in first-order predicate calculus), since it involves defeasible inferences (as we saw in the case of persistences). Eventually this problem with Hayes' theory must be addressed, and our research suggests that it can't be solved within the framework of present default logics.

If nonmonotonic logics are to be more than a mathematical curiosity, it must be demonstrated that they correctly represent some significant class of reasoning behavior falling outside the boundaries of traditional deduction. It was originally hoped that "default inference" was such a class: that there was a significant class of problems (temporal projection among them) in which the reasoning process was "deductive except in exceptional cases", and in which the exceptions to deductive reasoning could be expressed by the default rules.

It's not clear any more that such a class exists. While there has been limited success

in proving that default logics do the right thing (most notably the work of Etherington [7] and Etherington and Reiter [8] in formalizing inheritence hierarchies with exceptions), our demonstration that temporal projection is *not* reasoning of this same sort calls into question whether nonmonotonic logics are indeed adequate to represent a significant class of non-deductive reasoning problems.

If this is indeed the case—that is, if a significant part of defeasible reasoning can't be represented by default logics, and if in the cases where the logics fail we have no better way of describing the reasoning process than by a direct procedural characterization (like our program or its inductive definition), then logic as an AI representation language begins to look less and less attractive.

References

[1] Allen, James F., "Towards a General Theory of Action and Time", Computer Science Technical Report 97 (Revised), University of Rochester, October 1983.

[2] Davis, Martin, "The Mathematics of Non-Monotonic Reasoning", *Artificial Intelligence*, vol. 13 (1980), pp. 73–80.

[3] Dean, Tom, "Temporal Imagery: An Approach to Reasoning about Time for Planning and Problem Solving", Ph.D. Dissertation, Yale University Department of Computer Science, 1985.

[4] Doyle, Jon, "A Truth Maintenance System", *Artificial Intelligence*, vol. 12, no. 3 (1979), pp. 231–272.

[5] Doyle, Jon, "Non-Deductive Reasoning and Non-Monotonic Logics", in *Three Short Essays on Decisions, Reasons, and Logics*, Computer Science Technical Report STAN-CS-81-864, Stanford University, May 1981.

[6] Enderton, Herbert B., *A Mathematical Introduction to Logic*, Academic Press, 1972.

[7] Etherington, David W., "Formalizing Non-Monotonic Reasoning Systems", Computer Science Technical Report No. 83-1, University of British Columbia.

[8] Etherington, David W. and Raymond Reiter, "On Inheritance Hierarchies With Exceptions", *Proceedings AAAI-83*, pp. 104–108.

[9] Hayes, Patrick J., "Naive Physics I: An Ontology for Liquids", in Hobbs, ed., *Formal Theories of the Commonsense World*, 1985.

[10] Joshi, Aravind, Bonnie Webber and Ralph Weischedel, "Default Reasoning in Interaction", *Proceedings of the Non-Monotonic Reasoning Workshop*, AAAI, October 1984, pp. 151–164.

[11] Lifschitz, Vladimir, "Some Results on Circumscription", *Proceedings of the Non-Monotonic Reasoning Workshop*, AAAI, October 1984, pp. 151–164.

[12] McCarthy, John, "Circumscription – A Form of Non-Monotonic Reasoning", *Artificial Intelligence*, vol. 13 (1980), pp. 27–39.

[13] McCarthy, John, "Applications of Circumscription to Formalizing Common Sense Knowledge", *Proceedings of the Non-Monotonic Reasoning Workshop*, AAAI, October 1984, pp. 295–324.

[14] McCarthy, John, and P. J. Hayes, "Some philosophical problems from the standpoint of artificial intelligence", in: B. Meltzer and D. Michie (eds.), *Machine Intelligence 4*, Edinburgh University Press, 1969, pp. 463–502.

[15] McDermott, Drew V., "Nonmonotonic Logic II: Nonmonotonic Modal Theories", *JACM*, vol. 29, no. 1 (1982), pp. 33–57.

[16] McDermott, Drew V., "A Temporal Logic for Reasoning About Processes and Plans", *Cognitive Science*, vol. 6 (1982), pp. 101–155.

[17] McDermott, Drew V., *DUCK: A LISP-Based Deductive System*, Computer Science Technical Report no. 399, Yale University, June 1985.

[18] McDermott, Drew V., *The NISP MANUAL*, Computer Science Technical Report no. 274, Yale University, June 1983.

[19] McDermott, Drew V. and Jon Doyle, "Non-Monotonic Logic I", *Artificial Intelligence*, vol. 13 (1980), pp. 41–72.

[20] Moore, Robert C., "Reasoning About Knowledge and Action", Ph.D. Dissertation, MIT, 1979.

[21] Moore, Robert C., "Semantical Considerations on Nonmonotonic Logic", SRI Technical Note 284, June 1983.

[22] Perlis, Donald, and Jack Minker, "Completeness Results for Circumscription", Computer Science Technical Report TR-1517, University of Maryland.

[23] Reiter, Raymond, "A Logic for Default Reasoning", *Artificial Intelligence*, vol. 13 (1980), pp. 81–132.

[24] Reiter, Raymond, "Circumscription Implies Predicate Completion (Sometimes)", *Proceedings, AAAI-82*, pp. 418–420.

[25] Shoham, Yoav, "Time and Causation from the Standpoint of Aritificial Intelligence", Computer Science Research Report, Yale University, forthcoming (1985).

A Axioms for Describing Persistences and Clipping

Variables beginning with question mark are universally quantified. Constant symbols (axiom 2 only) are in upper case, and Skolem functions are written in "traditional" functional notation (*e.g.* `pt(pc-tok, evt-tok, ofact-tok)`).

Axiom 1 Creating persistence tokens

```
(if (and (pcause ?pc-tok ?ofact-pat ?evt-pat ?nfact-pat)
         (persist ?ofact-tok ?ofact-pat ?ofact-bp ?ofact-ep)
         (event ?evt-tok ?evt-pat ?evt-pt)
         (≤ ?ofact-bp ?evt-pt)
         (≤ ?evt-pt ?ofact-ep)))
    (and (persist
            pt(?pc-tok, ?evt-tok, ?ofact-tok)
            ?ofact-pat
            pb(?pc-tok, ?evt-tok, ?ofact-tok)
            pe(?pc-tok, ?evt-tok, ?ofact-tok))
         (~ pb(?pc-tok, ?evt-tok, ?ofact-tok)
            ?evt-pt)))
```

Axiom 2 The ALWAYS token

```
(persist PTA ALWAYS PBA PEA)

(if (point ?p)
    (or (= ?p PBA)
        (≺ PBA ?p1)))

(if (point ?p)
    (not (clipped PTA ?p1)))
```

Axiom 3 Persistence clipped directly by a contradictory token

```
(if (and (persist ?fact-tok ?fact-pat ?fact-bp ?fact-ep)
         (persist ?clip-tok ?clip-pat ?clip-bp ?clip-ep)
         (contradict ?fact-pat ?clip-pat)
         (≺ ?fact-bp ?clip-bp))
    (clipped ?fact-tok ?clip-bp))
```

Axiom 4 Persistence clipped indirectly by a point

```
(if (and (persist ?fact-tok ?fact-pat ?fact-bp ?fact-ep)
         (clipped ?fact-tok ?p1)
         (point ?p1)
         (point ?p2)
         (⪯ ?p1 ?p2))
    (clipped ?fact-tok ?p2))
```

Axiom 5 Clipping implies $\prec$

```
(if (and (persist ?tok ?pat ?bp ?ep)
         (clipped ?tok ?p1))
    (≺ ?ep ?p1))
```

Axiom 6 Persistences endure unless clipped

```
(if (and (persist ?fact-tok ?fact-pat ?fact-bp ?fact-ep)
         (point ?p1)
         (not (clipped ?fact-tok ?p1)))
    (⪯ ?p1 ?fact-ep))
```

Axiom 7 Begin point must come before end point

(if (persist ?tok ?pat ?p_1 ?p_2)
(≺ ?p_1 ?p_2))

Axiom 8 Defining what things are points

(if (persist ?tok ?pat ?p_1 ?p_2)
(and (point ?p_1)
(point ?p_2)))

(if (event ?tok ?pat ?p_1)
(point ?p_1))

Axiom 9 Contradiction is symmetric

(if (contradict ?pat_1 ?pat_2)
(contradict ?pat_2 ?pat_1))

For these point ordering axioms I will assume that variables starting with "?p" are of type *point*. That is, all axioms are of the form:

```
(if (and (point ?p1)
         (point ?p2)
         ...)
    -body-)
```

Axiom 10 $\sim$ is reflexive, symmetric, and transitive

```
(~ ?p1 ?p1)

(if (~ ?p1 ?p2)
    (~ ?p2 ?p1))

(if (and (~ ?p1 ?p2)
         (~ ?p2 ?p3))
    (~ ?p1 ?p3))
```

Axiom 11 $\prec$ is non-reflexive, antisymmetric, and transitive

```
(not (≺ ?p1 ?p1))

(if (≺ ?p1 ?p2)
    (not (≺ ?p2 ?p1)))

(if (and (≺ ?p1 ?p2)
         (≺ ?p2 ?p3))
    (≺ ?p1 ?p3))
```

Axiom 12 Transitivity relations for $\preceq$ and $\prec$

(if (and ($\preceq$?p_1 ?p_2)
($\preceq$?p_2 ?p_3))
($\preceq$?p_1 ?p_3))

(if (and ($\prec$?p_1 ?p_2)
($\preceq$?p_2 ?p_3))
($\prec$?p_1 ?p_3))

(if (and ($\preceq$?p_1 ?p_2)
($\prec$?p_2 ?p_3))
($\prec$?p_1 ?p_3))

Axiom 13 $\sim$ and $\prec$ imply $\preceq$

(if ($\sim$?p_1 ?p_2)
($\preceq$?p_1 ?p_2))

(if ($\prec$?p_1 ?p_2)
($\preceq$?p_1 ?p_2))

Axiom 14 $\sim$ and $\preceq$ incompatible with converse form of $\prec$

(if ($\sim$?p_1 ?p_2)
(not ($\prec$?p_2 ?p_1)))

(if ($\preceq$?p_1 ?p_2)
(not ($\prec$?p_2 ?p_1)))

Axiom 15 Substituting $\sim$ points preserves $\prec$ and $\preceq$

(if (and (~ ?p1 ?p2)
(≺ ?p1 ?p3))
(≺ ?p2 ?p3))

(if (and (~ ?p1 ?p2)
(≺ ?p3 ?p1))
(≺ ?p3 ?p2))

(if (and (~ ?p1 ?p2)
(⪯ ?p1 ?p3))
(⪯ ?p2 ?p3))

(if (and (~ ?p1 ?p2)
(⪯ ?p3 ?p1))
(⪯ ?p3 ?p2)))

B Program for Reasoning about Events and Facts

This appendix will be in two parts: a high-level description of the code, and a listing.

B.1 Program description

Refer to appendix C for an example of program input and output. The general purpose of the program is to take information about causality (in the form of `pcause` and `contradict` assertions) and about events occuring, and to output lists of persistences, points, clippings, and point orderings.

The program's state is contained, for the most part, in the set of assertions posted to the DUCK database, involving the relations `pcause`, `contradict`, `event`, `persist`, and `bucket`. The first four have meanings corresponding to predicates in the logic. `Bucket` is a relation over points, and is used to infer point orderings. It is explained below.

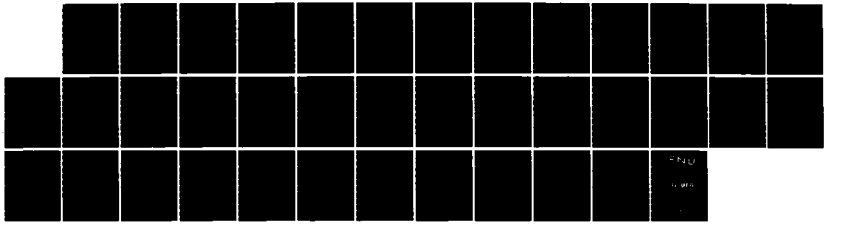
The main operations the program must perform are these:

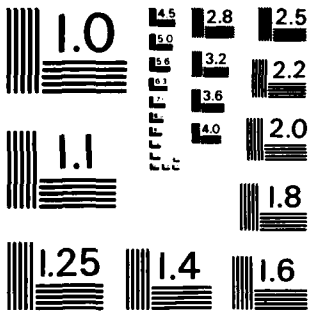
1. add a new event (function ***add-event***)
2. bring a new persistence into existence (functions ***event-cause-persist*** and ***persist-cause-persist***)
3. clip a persistence (function ***persist-clip-persist***)

Information about the events, persistences, pcauses, and contradictions correspond to explicit assertions in the DUCK database. (So to get a list of the persistences the program just fetches all DUCK assertions involving that predicate.) But points, point orderings, and clippings are represented implicitly. This information is held in DUCK assertions of the form `(bucket p n)`, where `p` is a point (that is, an event point or persistence begin or

AD-A162 447 TEMPORAL REASONING AND DEFAULT LOGICS(U) YALE UNIV NEW HAVEN CT DEPT OF COMPUTER SCIENCE S HANKS ET AL. OCT 85 YALEU/CSD/RR-430 N00014-85-K-0301 2/2

UNCLASSIFIED F/G 5/10 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

end point), and n is a nonnegative integer (representing the "bucket index" of the point). A lower bucket index generally means a point occuring earlier in time, but we'll see in a moment exactly what information is conveyed by the index.

When the program adds a new event it (conceptually) adds two new buckets, with indices one and two greater than those added for the last event. Into the first it puts persistence endpoints for those facts clipped directly by newly created facts ("newly created facts" being those caused by the event just added). Into the second bucket it puts the new event point, along with the begin points of any persistences caused by the new event. The i^{th} event point will be put in bucket $2i$; endpoints of persistences first clipped by the i^{th} event will be put in bucket $2i - 1$.

Aside from the $2n$ buckets created as a result of processing n events, there is a bucket (with index 0) holding the point PBA—the begin point of the ALWAYS fact—and a bucket holding endpoints of all persistences not currently clipped by *any* points. The "unclipped" bucket has index $2n + 1$. A clipped persistence, then, is one whose endpoint is *not* in the "unclipped" bucket, and the points that clip it are exactly those points with numerically greater bucket indices.

Determining point orderings, say between points p_1 and p_2, is a matter of retrieving the associated bucket assertions—for example (bucket p_1 b_1) and (bucket p_2 b_2). The rules for assigning an ordering for points p_1 and p_2, given bucket indices b_1 and b_2 respectively, are as follows:

1. if $p_1 = p_2$ then ($\sim$ p_1 p_2)
2. if $b_1 = b_2$ and b_1 is even, then ($\sim$ p_1 p_2)
3. if $b_1 < b_2$ then ($\prec$ p_1 p_2)

4. if $b_1 \leq b_2$ then $(\preceq\ p_1\ p_2)$.

Note that $\preceq$ doesn't strictly mean "either $\sim$ or $\prec$". If p_1 and p_2 are both in bucket $2n + 1$, for example, rule 2 dictates that $(\preceq\ p_1\ p_2)$ and $(\preceq\ p_2\ p_1)$, but not $(\sim\ p_1\ p_2)$.

In appendix E we prove that these rules lead to the correct transitivity, symmetry, *etc.* properties.

Now we present the program code itself.

B.2 Program code

```
;*****************************************************************
; 6/9/85
;
;  Simple temporal program for events, persistences, and time-point
;  ordering.   Most all the work is done by DUCK assertions of the
;  form EVENT, PERSISTENCE, and BUCKET
;

;  Corresponding to terms in the logic are TOKENs (unique identifiers
;    for PCAUSEs, EVENTs, and PERSISTences), PATTERNs (the fact type
;    asserted by a particular event or persistence), and POINTs (points
;    in time that also define intervals over which persistences hold).

(deftype TOKEN  SEXP)
(deftype PATTERN SEXP)
(deftype POINT   SEXP)

(specdecl INTEGER
   (eventno* 1)          ; for making up TOKEN names for events and
   (pcauseno* 1))        ; pcause assertions


(specdecl FIXNUM  (debug-level* 1))
    ;  0 is none, 1 is cursory, >1 is verbose

(:= type-check* ())
    ;  keep NISP from complaining about DUCK for-each-ans bindings


;*****************************************************************
;  The main thing we do is add a new event -- only its PATTERN is
;  input here, and the event is assumed to happen AFTER the last event
;  that was input.  Here all we do is make up a (unique) token name for
;  the event and for the time point at which it occurs, ADD it to the
;  DUCK database, and see what persistences it causes.

(proc add-event VOID (PATTERN event-pat)
  (let  (TOKEN  (event-tok (symbol EVT  - (< eventno*))))
        POINT  (event-pt  (symbol PEVT - (< eventno*))))
    (:= eventno* (+ *-* 1))
```

```
    (make-evt-buckets event-pt)
    (add (list 'event event-tok event-pat event-pt))))

(lisprule event (event ?tok ?pat ?b) ->
  (evt-cause-persist ?tok ?pat ?b))

;*************************************************************************
; EVT-CAUSE-PERSIST and PERSIST-CAUSE-PERSIST are pretty much the same:
;  check the database to see whether there are PCAUSE, PERSIST, and
;  EVENT assertions to license adding a new persistence, and if so,
;  add it.

(proc evt-cause-persist VOID (TOKEN evt-tok PATTERN evt-pat POINT evt-bp)
  (for-each-ans
     (fetch ¡(pcause ?pcause-tok ?fact-pat ,evt-pat ?der-pat))
    (for-each-ans
       (fetch ¡(persist ?fact-tok ,?fact-pat ?fact-bp ?fact-ep))
          (cond
            ((in-interval evt-bp ?fact-bp ?fact-ep)
               (add-persist ?der-pat ?pcause-tok evt-tok ?fact-tok))))))

(proc persist-cause-persist VOID (TOKEN fact-tok PATTERN fact-pat POINT bf ef)
  (for-each-ans (fetch ¡(pcause ?pcause-tok ,fact-pat ?evt-pat ?der-pat))
    (for-each-ans (fetch ¡(event ?evt-tok ,?evt-pat ?be))
      (cond
        ((in-interval ?be bf ef)
           (add-persist ?der-pat ?pcause-tok ?evt-tok fact-tok))))))

; ADD-PERSIST ADDs a new persistence with the given pattern to the
; database.  It makes up "Skolem" terms for the persistence's token-id,
; begin point, and end point,  then adds the PERSIST assertion and the
; POINT assertions to the database.  Doing the ADD will automatically
; trigger calls to PERSIST-CLIP-PERSIST and PERSIST-CAUSE-PERSIST as well.

(proc add-persist VOID (PATTERN der-pat TOKEN pcause-tok evt-tok fact-tok)
  (let ((names (list pcause-tok evt-tok fact-tok)))
     (add-point (cons 'pb names) 'event)
     (add-point (cons 'pe names) 'persist)
     (add (list 'persist
                (cons 'pt names)
                der-pat
                (cons 'pb names)
```

```
                (cons 'pe names)))))

(lisprule chain (persist ?tok ?pat ?b ?e) ->
  (persist-clip-persist ?tok ?pat ?b ?e)
  (persist-cause-persist ?tok ?pat ?b ?e))


; PERSIST-CLIP-PERSIST looks to see if the given persistence named
; by the given tokenid may clip any existing persistences.  To do so
; we just look in the database for any contradictory patterns, see
; if there are any tokens that assert these patterns, and see if they
; happen in time before the input persistence.  If so, all there is
; to clipping is to move the clipped persistence's endpoint before
; the current event point.  There's also a "consistency" check, to
; prevent the situation where a persistence knocks out its own
; support (in other words, where a persistence gets clipped by the
; same event that caused it)

(proc persist-clip-persist VOID (TOKEN tokid PATTERN pat POINT bp ep)
  (for-each-ans (fetch ¡(contradict ?clipped-pat ,pat))
    (for-each-ans (fetch ¡(persist ?clipped-tok ,?clipped-pat ?bc ?ec))
      (cond
        ((pteq bp ?bc)
          (earror persist-clip-persist
                  ()
                  "Inconsistency -- token " ?clipped-tok " trying to clip "
                    tokid -1))
        ((in-interval bp ?bc ?ec)
          (delete-point ?ec)
          (add-point ?ec 'clip))))))

;*****************************************************************
;  Manipulating points and orderings.
;    All points are put into buckets when the corresponding event or
;    persistence is added to the database.  Buckets are represented by
;    a DUCK assertion of the form (bucket p b) where b is an integer.
;    In general a smaller bucket means earlier in time.  Bucket number one
;    contains just the begin point of the ALWAYS persistence.  Odd-numbered
;    buckets contain only event points and persistence begin points;
;    even-numbered buckets contain only persistence end points.  End points
;    of unclipped persistences are given a bucket number of PERSIST-BUCK*
;    --  2n+1 where n is the number of events.
```

```
(specdecl FIXNUM
          (persist-buck* 0)      ; this will be set by init-buckets
          (cur-evt-buck* 0))     ; incremented by 2 on every event

; BUCKET function maps a point into its bucket number

(func bucket INTEGER (POINT p)
  (let ((f (fetch ¡(bucket ,p ?x))))
    ?(x (car f))))

; A bucket index refers to an event bucket iff it's even

(func event-bucket BOOLEAN (FIXNUM buck-indx)
  (= buck-indx (* 2 (fix (/ buck-indx 2)))))


; ADD-POINT -- add point to a bucket (by making appropriate DUCK
;  assertion.)  Second argument is one of:
;      'EVENT         add the most recent event bucket
;      'CLIP          add to the most recent clipped bucket
;      'PERSIST       add to the bucket of unclipped persistence endpoints

(proc add-point VOID (POINT pt OBJ where)
  (selq where
     ((event)   (add ¡(bucket ,pt ,cur-evt-buck*)))
     ((clip)    (add ¡(bucket ,pt ,(- cur-evt-buck* 1))))
     ((persist) (add ¡(bucket ,pt ,persist-buck*)))
     (t         (earror add-point () "invalid add option:  " where))))

(proc delete-point VOID (POINT pt)
   (for-each-ans (fetch ¡(bucket ,pt ?x))
     (erase ¡(bucket ,pt ?x))))


;
;  Point orderings --  =p <p =<p
;    Two points are =p if they're the same point, or if they're in
;     the same event bucket.
;    (<p p1 p2) if their respective buckets are so ordered by <
;    (=<p p1 p2) if their respective buckets are so ordered by <=

(func =p BOOLEAN (POINT p1 p2)
```

```
  (let ((b1 (bucket p1)) (b2 (bucket p2)))
    (or (pteq p1 p2)
        (and (= b1 b2) (event-bucket b1)))))

(func <p BOOLEAN (POINT p1 p2)
  (let ((b1 (bucket p1)) (b2 (bucket p2)))
    (< b1 b2)))

(func =<p BOOLEAN (POINT p1 p2)
  (let ((b1 (bucket p1)) (b2 (bucket p2)))
    (<=  b1 b2)))

; IN-INTERVAL -- Does p1 fall within the interval formed by p2 and p3?

(func in-interval BOOLEAN (POINT p1 p2 p3)
  (and (=<p p2 p1) (=<p p1 p3)))

; PTEQ -- Are p1 and p2 identical points?

(func pteq BOOLEAN (POINT p1 p2)
  (cond
    ((and (quark p1) (quark p2)) (eq p1 p2))
    ((or  (quark p1) (quark p2))                ())
    (t   (and (eq (car p1) (car p2))
              (<& pteq (cdr p1) (cdr p2))))))

(func ptindx FIXNUM (POINT p (lst POINT) plist)
  (let (FIXNUM (fail (+ -1 (* -1 (length plist))))
               (ans -1))
    (:= ans (ptindx-aux p plist fail))
    (cond ((> ans 0)    ans)
          (t             -1))))

(func ptindx-aux FIXNUM (POINT p (lst POINT) plist FIXNUM fail)
  (cond
    ((null plist)          fail)
    ((pteq p (car plist))   1)
    (t  (+ 1 (ptindx-aux p (cdr plist) fail)))))

; MAKE-EVT-BUCKETS -- move the pointer CUR-EVT-BUCK* up by two, to
;  reflect a new event bucket and a bucket for those persistence endpoints
;  clipped by the new event point.  Add point P to the new event bucket.
```

```
(proc make-evt-buckets VOID (POINT p)
  (:= cur-evt-buck* (+ 2 *-*))
  (add-point p 'event))

(proc init-buckets VOID (FIXNUM n POINT bp ep)    ; n is number of events
  (:= cur-evt-buck* -2)  ; this will immediately be bumped to 0
  (make-evt-buckets bp)
  (:= persist-buck* (+ (* 2 n) 1))
  (add-point ep 'persist))

;*************************************************************
; User interface


; INIT -- erase everything, and add the ALWAYS token

(proc init VOID (FIXNUM n)    ; n is number of events
  (erase-assertions)
  (init-buckets n 'PBA 'PEA)
  (add '(persist PTA ALWAYS PBA PEA)))

(proc erase-assertions VOID ()
  (for-each-ans (fetch '(bucket ?p ?b))
    (erase '(bucket ?p ?b)))
  (for-each-ans (fetch '(persist ?tok ?pat ?beg ?end))
    (erase '(persist ?tok ?pat ?beg ?end)))
  (for-each-ans (fetch '(event ?tok ?pat ?beg))
    (erase '(event ?tok ?pat ?beg)))
  (for-each-ans (fetch '(contradict ?pat1 ?pat2))
    (erase '(contradict ?pat1 ?pat2)))
  (for-each-ans (fetch '(pcause ?tok ?fpat ?epat ?dpat))
    (erase '(pcause ?tok ?fpat ?epat ?dpat))))

;*************************************************************************
; Running a problem involves initializing, adding contrdiction and
;  pcause assertions, then adding the events one by one.  Then we
;  dump out the results.  Input to RUN-TEST is a list of contradictions,
;  a list of pcauses, and a list of events.

(proc run-test VOID (ipt)
   (init (length (caddr ipt)))
```

```
   (stdmsg -1 "**********" -1)
   (add-contras (car ipt))
   (add-pcauses (cadr ipt))
   (add-events  (caddr ipt))
   (dump-inputs)
   (dump-outputs)
   (stdmsg -1 "**********"))

(proc add-contras VOID (contras-in)
  (loop for (OBJ (contra-clause in contras-in))
    (add (cons 'contradict contra-clause))))

(proc add-pcauses VOID (pcauses-in)
  (loop for (OBJ (pcause-clause in pcauses-in))
    (add (cons 'pcause
               (cons (symbol PC - (++ pcauseno*))
                     pcause-clause)))))

(proc add-events VOID (events-in)
  (loop for ((e in events-in))
    (add-event e)))

;*****************************************************************************
; Printing results:  dumping inputs means all pcauses, events and contras.
;   for output we dump persistences, points, and clippings, and point orders.

(proc dump-inputs VOID ()
  (stdmsg -1 "Pcauses:" -1)
  (for-each-ans (fetch '(pcause ?tt ?fpat ?epat ?dpat))
     (stdmsg (t 4) ?tt (t 12) ?fpat (t 20) ?epat (t 28) ?dpat -1))
  (stdmsg -1 "Contradictions:" -1)
  (for-each-ans (fetch '(contradict ?pat1 ?pat2))
     (stdmsg (t 4) ?pat1 (t 28) ?pat2 -1))
  (stdmsg -1 "Events:" -1 )
  (for-each-ans (fetch '(event ?tt ?pat ?b))
    (stdmsg (t 4) ?tt (t 12) ?pat (t 20) ?b -1)))

(func dump-outputs VOID ()
  (let ((persist-list (build-persist-list))
        (point-list   (build-point-list)))
    (dump-persists persist-list)
    (dump-points point-list)
```

```
    (dump-clips persist-list point-list)
    (dump-orderings point-list)))

(func dump-persists VOID (persist-list)
  (stdmsg -2 "Persistences:" -1 )
  (loop for ((persist in persist-list)
             FIXNUM (i 1 (+ i 1)))
     (stdmsg (t 4) i "."
             (t 8)  (car persist)    -1
             (t 10) (cadr persist)   -1
             (t 10) (caddr persist)  -1
             (t 10) (cadddr persist) -1)))

(func dump-points VOID (point-list)
  (stdmsg -2 "Points:" -1)
  (loop for ((pt in point-list)
             FIXNUM (i 1 (+ i 1)))
     (stdmsg (t 5) i "." (t 9) pt -1)))

(func dump-clips VOID (persist-list point-list)
  (stdmsg -2 "Clippings:" -1 (t 4) "-- persist --" (t 20) "-- point --" -1)
  (loop for ((persist in persist-list)
             FIXNUM (per-index 1 (+ per-index 1))
             POINT  (endpt ())
             FIXNUM (endbuck ()))
      (:= endpt (cadddr persist))
      (:= endbuck (bucket endpt))
      (loop for (POINT (pt in point-list))
        (cond
          ((< endbuck (bucket pt))
            (stdmsg (t 10) per-index (t 25) (ptindx pt point-list) -1))))))

(func dump-orderings VOID (point-list)
  (let (FIXNUM (n (length point-list)))
    (stdmsg -2 "Orderings:" -1)
    (loop for  ((i = 1 to n))
      (stdtab (* 4 i)) (stddisplay i))
    (stdnewline)
    (loop for  ((i = 1 to n))
      (stddisplay i)
      (loop for ((j = 1 to n))
        (stdtab (* 4 j))
```

```
        (stddisplay (ordsym (nthelem i point-list) (nthelem j point-list))))
      (stdnewline))))


(func build-persist-list (lst OBJ) ()
  (let ((res ()))
    (for-each-ans (fetch '(persist ?tok ?pat ?bp ?ep))
      (:= res (cons (list ?tok ?pat ?bp ?ep) *-*)))
    res))

(proc build-point-list (lst POINT) ()
  (let ((pts ())
        (n 0))
    (for-each-ans (fetch '(bucket ?p ?b))
      (:= pts (insert-by-buck ?p ?b *-*)))
    (:= pts (<# car *-*))
    pts))


; Insert point P with bucket index B into list PTS, maintaining the
; list in ascending order of bucket index.   PTS is a list of
; (point . bucket) pairs.

(proc insert-by-buck OBJ (POINT p FIXNUM b OBJ pts)
  (cond ((null pts)
         (list (cons p b)))
        ((<= b (cdr (car pts)))
         (cons (cons p b) pts))
        (t
         (cons (car pts) (insert-by-buck p b (cdr pts))))))

(func ordsym SYMBOL (POINT pti ptj)
    (cond
      ((pteq pti ptj)  '=)
      ((<p pti ptj)    '<)
      ((=p pti ptj)    '=)
      ((=<p pti ptj)   '<=)
      ((<p ptj pti)    '>)
      ((=<p ptj pti)   '>=)
      (t               'X)))

(func flatten (lst POINT) ((lst (lst POINT)) pts)
```

```
  (cond
    ((null pts)    ())
    (t  (append (car pts) (flatten (cdr pts))))))

;*************************************************************
;  Run a test into a named transcript file

(func make-tscript STRING (STRING filename OBJ test-data)
  (let ((str (openo filename))
        (std (stdout)))
    (stdout-set str)
    (run-test test-data)
    (stdout-set std)
    "DONE"))

;*************************************************************
;  Test cases

(:= test-1
    (list     ; contradtions
          '(((alive ?x) (dead ?x)))
              ; PCAUSEs
          '((ALWAYS      (born ?x)            (alive ?x))
            (ALWAYS      (load ?g)            (loaded ?g))
            ((loaded ?g) (shoot ?x ?g)        (dead ?x))
            ((alive ?x)  (win-sweepstakes ?x) (rich ?x)))
              ; EVENTs
          '((born JOHN)
            (load GUN)
            (shoot JOHN GUN)
            (win-sweepstakes JOHN))))

(make-tscript "pertest.tex" test-1)
```

C Program Input and Output

Input

```
(list
          ; CONTRADICTIONs
      '(((alive ?x) (dead ?x)))

          ; PCAUSEs
      '((ALWAYS      (born ?x)            (alive ?x))
        (ALWAYS      (load ?g)            (loaded ?g))
        ((loaded ?g) (shoot ?x ?g)        (dead ?x))
        ((alive ?x)  (win-sweepstakes ?x) (rich ?x)))

          ; EVENTs
      '((born JOHN)
        (load GUN)
        (shoot JOHN GUN)
        (win-sweepstakes JOHN)))
```

Output

```
Pcauses:
    PC-5    (alive ?3)
              (win-sweepstakes ?3)
              (rich ?3)
    PC-4    (loaded ?5)
              (shoot ?4 ?5)
              (dead ?4)
    PC-3    ALWAYS
              (load ?6)
              (loaded ?6)
    PC-2    ALWAYS
              (born ?7)
              (alive ?7)

Contradictions:
    (alive ?8)        (dead ?8)

Events:
    EVT-4   (win-sweepstakes JOHN)
                    PEVT-4
    EVT-3   (shoot JOHN GUN)
                    PEVT-3
    EVT-2   (load GUN)
                    PEVT-2
    EVT-1   (born JOHN)
                    PEVT-1


Persistences:
    1.  PTA
          ALWAYS
          PBA
          PEA
    2.  (pt PC-4 EVT-3 (pt PC-3 EVT-2 PTA))
          (dead JOHN)
          (pb PC-4 EVT-3 (pt PC-3 EVT-2 PTA))
          (pe PC-4 EVT-3 (pt PC-3 EVT-2 PTA))
    3.  (pt PC-3 EVT-2 PTA)
          (loaded GUN)
          (pb PC-3 EVT-2 PTA)
```

```
      (pe PC-3 EVT-2 PTA)
4.  (pt PC-2 EVT-1 PTA)
      (alive JOHN)
      (pb PC-2 EVT-1 PTA)
      (pe PC-2 EVT-1 PTA)


Points:
    1.  PBA
    2.  PEVT-1
    3.  (pb PC-2 EVT-1 PTA)
    4.  PEVT-2
    5.  (pb PC-3 EVT-2 PTA)
    6.  (pe PC-2 EVT-1 PTA)
    7.  PEVT-3
    8.  (pb PC-4 EVT-3 (pt PC-3 EVT-2 PTA))
    9.  PEVT-4
    10. PEA
    11. (pe PC-3 EVT-2 PTA)
    12. (pe PC-4 EVT-3 (pt PC-3 EVT-2 PTA))


Clippings:
   -- persist --   -- point --
        4              7
        4              8
        4              9
        4              10
        4              11
        4              12
```

Orderings:

	1	2	3	4	5	6	7	8	9	10	11	12
1	=	<	<	<	<	<	<	<	<	<	<	<
2	>	=	=	<	<	<	<	<	<	<	<	<
3	>	=	=	<	<	<	<	<	<	<	<	<
4	>	>	>	=	=	<=	<	<	<	<	<	<
5	>	>	>	=	=	<=	<	<	<	<	<	<
6	>	>	>	>=	>=	=	<	<	<	<	<	<
7	>	>	>	>	>	>	=	=	<	<	<	<
8	>	>	>	>	>	>	=	=	<	<	<	<
9	>	>	>	>	>	>	>	>	=	<=	<=	<=
10	>	>	>	>	>	>	>	>	>=	=	<=	<=
11	>	>	>	>	>	>	>	>	>=	<=	=	<=
12	>	>	>	>	>	>	>	>	>=	<=	<=	=

D Inductive Description of Algorithm Output.

Herein we describe the program as a series of stages, the i^{th} stage representing the program's state after adding the i^{th} event to its database. The description is a set of rules, one for each of the relations corresponding to the temporal predicates, along with rules specifying what changes to the relation are allowed in transition from the i^{th} to the $i+1^{st}$ stage.

Since the description is inductive in the number of events, each relation name will be subscripted by an event index. The relation $\texttt{clipped}_i$, for example, represents the program's representation of the relation `clipped` after i events have been processed. Assuming there are n such events, the program's output is described by $\texttt{pcause}_n$, $\texttt{persist}_n$, $\preceq_n$, *etc.*

We must therefore demonstrate that the description *actually does* characterize the program *correctly. To this end we must show three things for each relation* $\mathcal{R}$*:*

1. that the initial contents of $\mathcal{R}$ in the program is the same as the description's contents of $\mathcal{R}$ at stage 0,
2. that the individuals added or deleted by the program in adding the $i+1^{st}$ event are the same as described for the transition between the description's i^{th} and $i+1^{st}$ stages for that relation, and
3. that these initial and transitional additions and deletions are the *only* changes made by the program to that relation. (In other words we have to verify that the last item in each relation description, which says "no others", is indeed true.)

D.1 Notation

We will use the following notational shorthand:

1. If $\mathcal{R}$ is a relation, then $\vec{x} \in \mathrm{R}_{i+1\setminus i}$ stands for $\vec{x} \in (\mathrm{R}_{i+1} \setminus \mathrm{R}_i)$. In terms of the program, this means "$\vec{x}$ is added to relation $\mathcal{R}$ in the process of adding the $i+1^{st}$ event."

2. Saying "the end point of tok is pe", abbreviates "for some i, pat and pb: (tok, pat, pb, pe) $\in$ persist$_i$". The same holds for the begin point of a token.

3. Saying "tok is first clipped at stage i" means that there is some p $\in$ point$_i$ such that (tok, p) $\in$ clipped$_i$, and that for any $j < i$ there is *no* point p' such that (tok, p') $\in$ clipped$_j$.

D.2 Input relations

Input to the program will be represented here as three sets, for pcauses, contradictions, and events:

1. The set **EVT** is the set corresponding to the *event* axioms in **P**. It is a set of n triples, each of the form (evt-tok$_i$, evt-pat$_i$, pevt$_i$). All evt-pat$_i$ are closed wffs.

2. The set **PC** corresponds to *pcause* axioms. These are tuples of the form (pc-tok$_j$, pc-ofact-pat($\vec{x}$), pc-evt-pat($\vec{x}$), pc-nfact-pat($\vec{x}$)), where pc-ofact-pat, pc-evt-pat, and pc-nfact-pat are terms free in the variables $\vec{x}$.

3. The set **CONTRA** corresponds to *contradict* axioms. These are pairs of the form (pat1($\vec{x}$), pat2($\vec{x}$)), where pat1 and pat2 are terms free in the variables $\vec{x}$.

The program stores event, pcause, and contradict assertions explicitly in the DUCK database. Furthermore, for the latter two relations these assertions are added at initialization and individuals are neither added nor deleted thereafter. Therefore we can describe pcause and contradict as follows:

D.2.1 Definition of pcause

1. if (pc-tok, pc-ofact-pat, pc-event-pat, pc-nfact-pat) $\in$ **PC**
   then (pc-tok, pc-ofact-pat, pc-event-pat, pc-nfact-pat) $\in$ pcause$_i$ $(i = 0, \ldots, n)$.

2. no others

D.2.2 Definition of contradict

1. if (pat1, pat2) $\in$ CONTRA
   then (pat1, pat2) $\in$ contradict$_i$ $(i = 0, \ldots, n)$.
2. no others

D.2.3 Definition of event

The definition of event is similar, but recall that the ith event assertion is added at the description's ith stage.

1. $event_0 = \phi$
2. $event_i \subseteq event_{i+1}$
3. (evt-tok$_{i+1}$, evt-pat$_{i+1}$, pevt$_{i+1}$) $\in$ event$_{i+1 \setminus i}$
4. no others

D.3 The persist relation

Persistences are stored as DUCK assertions of the form (persist tok pat bpoint epoint). We thus need to examine where in the code a DUCK *ADD* operation of this form takes place. There are two such places: in the function init, where the ALWAYS persistence is added, and in the function *add-persist*. Here is the inductive definition of persist (the relation unclipped-toks is defined on page 103):

D.3.1 Definition of persist

1. persist$_0$ = {(PTA, ALWAYS, PBA, PEA)}
2. persist$_i \subseteq$ persist$_{i+1}$

3. if (pc-tok, pc-ofact-pat, pc-evt-pat, pc-nfact-pat) $\in$ PC
   and there's a token *ofact-tok* with pattern *ofact-pat*,
   and a substitution σ such that (pc-evt-pat)σ = evt-pat$_{i+1}$,
   and (pc-ofact-pat)σ = ofact-pat,
   and ofact-tok $\in$ unclipped-toks$_i$;
   then (pt(pc-tok, evt-tok$_{i+1}$, ofact-tok),
        (nfact-pat)σ,
        pb(pc-tok, evt-tok$_{i+1}$, ofact-tok),
        pe(pc-tok, evt-tok$_{i+1}$, ofact-tok)) $\in$ persist$_{i+1\backslash i}$.

4. no others

Items 1 and 2 are straightforward: the ALWAYS tuple is added only once at initialization, and persist assertions are never retracted. It remains to show that item 3 corresponds to exactly those circumstances under which the function ***add-persist*** is called, and that the form of the persist assertion added agrees in form with that of the tuple added in item 3.

Note in the code that ***add-persist*** is called by ***persist-cause-persist*** and by ***event-cause-persist*** under similar circumstances: in each case the DUCK database is asked if there are pcause, event, and persist assertions corresponding to the first three preconditions of item 3. If there are, the program calls ***add-persist*** if the call to (boolean) function ***(in-interval pevt$_{i+1}$ ofact-bp ofact-ep)*** returns true—we have to verify that this will be the case exactly when the persistence named by ofact-tok is in the relation unclipped-toks$_i$.

The function *in-interval* in this case abbreviates the conjunction

```
(and (≤ ofact-bp pevt_{i+1})
     (≤ pevt_{i+1} ofact-ep).
```

The first conjunct is immediately verified: because the current event is added to new (later) buckets than any previous points, it's the case that for any persistence in persist$_{i+1}$, either

($\prec$ bp $\texttt{pevt}_{i+1}$) or ($\sim$ bp $\texttt{pevt}_{i+1}$). So in either case ($\preceq$ bp $\texttt{pevt}_{i+1}$). Also note that if $\texttt{ofact-tok} \in \texttt{unclipped-toks}_{i+1}$ then ($\preceq$ $\texttt{pevt}_{i+1}$ ofact-ep) directly by item 4 of the definition for $\preceq$.

It remains to show that if ($\preceq$ $\texttt{pevt}_{i+1}$ ofact-ep) then $\texttt{ofact-tok} \in \texttt{unclipped-toks}_i$. Assume $\texttt{ofact-tok} \notin \texttt{unclipped-toks}_i$, so by definition (ofact-tok, p) $\in \texttt{clipped}_{i+1}$ for some p. But by definition of $\prec$, it then follows that (ofact-ep, $\texttt{pevt}_{i+1}$) $\in \prec_{i+1}$. It is impossible, given the definition of $\prec$ and $\preceq$, for ($\preceq$ $\texttt{pevt}_{i+1}$ ofact-ep) to be true as well.

D.4 Point-defining relations

The program represents "pointhood" implicitly. Anything that has a bucket assertion is considered to be a point: all event points and persistence begin and end points.

D.4.1 Definition of point

1. $point_0 = \{PBA, PEA\}$
2. $point_i \subseteq point_{i+1}$
3. $\texttt{pevt}_{i+1} \in \texttt{point}_{i+1 \setminus i}$
4. if (tt, pat, pb, pe) $\in \texttt{persist}_{i+1 \setminus i}$
   then pb $\in \texttt{point}_{i+1 \setminus i}$
   and pe $\in \texttt{point}_{i+1 \setminus i}$
5. no others

For verification we need look at the places in the program where **add-point** is called: at initialization time to put PBA and PEA in buckets, and from **add-event** to add the current event point, and from **add-persist** to add the begin and end points of a newly-created

persistence. These cases correspond exactly to the items above.[4]

D.5 Clipping relations

Recall that clipping is represented indirectly—a persistence tok is clipped by a point (according to the program) just in case the end point of tok is in a bucket with a lower index than that of p. Note in the code that when a persistence is added to the database its end point is put in the "unclipped" bucket (the bucket with index $2n + 1$). Since this bucket has the highest possible index, persistences are at first unclipped by any point. Points are moved *out* of this bucket by the function ***delete-point***, which is called only by ***persist-clip-persist***.

The relations **clipped-toks**, **unclipped-toks**, and **unclipped-endpoints** are introduced just for notational convenience. Respectively they are the set of all tokens naming persistences clipped (by any point) at the i^{th} stage or before, those unclipped by any point as of the i^{th} stage, and the end points of those tokens in **unclipped-toks**.

D.5.1 Definition of clipped-toks

1. if there is a point p such that (tok, p) $\in$ clipped$_{i+1}$
   then tok $\in$ clipped-toks$_{i+1}$

D.5.2 Definition of unclipped-toks

1. if there is no point p such that (tok, p) $\in$ clipped$_{i+1}$
   then tok $\in$ unclipped-toks$_{i+1}$

[4] It's also the case that ***add-point*** is called when a persistence is clipped—but the same point is first deleted from a bucket then added to a different one, so no new point is created.

D.5.3 Definition of unclipped-endpoints

1. if tok $\in$ unclipped-toks$_{i+1}$
   and pe is the end point of tok
   then pe $\in$ unclipped-toks$_{i+1}$

Note that the points in unclipped-endpoints$_{i+1}$ are exactly those points in bucket $2n+1$.

D.5.4 Definition of clipped

1. $clipped_0 = \phi$
2. $clipped_i \subseteq clipped_{i+1}$
3. if tok$_1$, asserting pat$_1$ $\in$ unclipped-toks$_i$
   and there's a $tok_2 \in persist_{i+1\backslash i}$ asserting a pat$_2$
   and patterns (cpat$_1$, cpat$_2$) $\in$ contradict$_{i+1}$
   and a substitution σ such that
   $(cpat_1)\sigma = pat_1$ and $(cpat_2)\sigma = pat_2$
   and pb$_2$ is the begin point of tok$_2$
   then (tok$_1$, pb$_2$) $\in$ clipped$_{i+1\backslash i}$
4. if $p \in (point_{i+1\backslash i} \cup unclipped\text{-}endpoints_i)$
   and p is not the endpoint of a persistence clipped by item 3 above
   and tok $\in$ clipped$_i$
   or tok was clipped by item 3 above
   then (tok, p) $\in$ clipped$_{i+1\backslash i}$.
5. no others.

Item 1 is true because the ALWAYS persistence is the only persistence initially; its endpoint is put in the "unclipped" bucket, hence the token is unclipped.

Item 2 is true because once a persistence endpoint is move into a bucket other than bucket $2n+1$, no point can be moved into a bucket with a smaller index. (***Add-point*** can put points only into the current event bucket, the "clipped" bucket for the current event, and the ***persist*** bucket.)

Item 3: a "first clipping" at the i+1st stage is handled by ***persist-clip-persist***, and is similar to the persistence-causing situation we saw in defining **persist** above. As was the case with **persist**, the DUCK unification algorithm assures us of all the preconditions, except that we have to verify that $\mathtt{tok}_1 \in \mathtt{unclipped\text{-}toks}_i$ is equivalent to the program's condition *(in-interval* pb_2 pb_1 pe_1*)*. First of all, it must be the case that $(\preceq\ \mathrm{pb}_1\ \mathrm{pb}_2)$ since $\mathtt{tok}_2$ was created at the i+1st stage. Furthermore, since $\mathtt{tok}_1$ is in $\mathtt{unclipped\text{-}toks}_i$, pe_1 is in bucket $2n + 1$ as of the i+1st stage, thus has a bucket index greater than that of pb_2, and $(\preceq\ \mathrm{pb}_2\ \mathrm{pe}_1)$.[5]

Item 4: the set of points p described in item 4 are those points whose bucket index equal $2i + 2$ (*i.e.*, current event point and begin points of persistences caused at stage $i + 1$), or whose index is $2n + 1$ (the endpoints of persistences unclipped as of the end of the ith stage). The set of tokens tok are exactly whose endpoints are in buckets $2i + 1$ (clipped at the i+1st stage) or prior (clipped prior to the i+1st stage). This is exactly the set of points p and token endpoints pe such that the index of p is greater than the index of pe. By definition of **clipped**, these are the clipped tokens and the points that clip them.

D.6 Point-ordering relations

Our definition of the point-ordering relations will depart somewhat from the strictly inductive approach we have taken so far. The program computes point orderings by making an assertion of the form **(bucket p n)** for each point p, where n is a nonnegative integer, then computing point ordering based on points' respective bucket indices. We will do likewise:

[5]It may be the case that *two* tokens satisfy the preconditions for clipping a particular persistence. The program will choose one arbitrarily, and clip it according to the rules in item 3. The begin point of the *other* token will also clip it, but by item 4 instead.

first we provide an inductive definition of the relation bucket, then an algorithm to compute point orderings on the basis of the contents of that relation.

D.6.1 Definition of bucket

1. $\text{bucket}_0 = \{(\text{PBA}, 0), (\text{PEA}, 2n+1)\}$
2. $(\text{pevt}_i, 2i) \in \text{bucket}_i$
3. **if tok is a persistence created at stage i**
   and pb is the begin point of tok,
   and pe is the end point of tok,
   then $(\text{pb}, 2i) \in \text{bucket}_i$
   and $(\text{pe}, 2n+1) \in \text{bucket}_i$
4. **if tok is first clipped at the stage i,**
   and pe is the end point of tok
   then $(\text{pe}, 2i-1) \in \text{bucket}_i$
5. **otherwise, if** $\text{p} \in \text{point}_{i-1}$,
   and p is not the endpoint of a persistence first clipped at stage i
   and $(\text{p}, \text{n}) \in \text{bucket}_{i-1}$
   then $(\text{p}, \text{n}) \in \text{bucket}_i$

Item 1 is satisfied through function **init**; for items 2, 3, and 4, the program uses the variable ***eventno****, which is initialized to 0 and incremented by 2 every time an event is added. The current event point and new persistence begin points are added to bucket ***eventno****, and persistences clipped by that point are added to bucket $(eventno* - 1)$. Item 4 is verified by noting that *bucket* assertions are deleted only when a persistence is clipped, and this is the case of item 4. Item 5: bucket assertions are retracted only when a "first clipping" takes place (item 4). Otherwise assertions endure from stage to stage.

The first thing to notice about the definition of buckets is that every point is assigned exactly one bucket: at the 0^{th} stage the only two points (PBA and PEA) are both assigned

buckets, and all points created at the i+1st stage are assigned buckets either by item 2 or item 3. If p is an event point or a persistence begin point, the bucket index it receives on creation (which is $2i$ if it's created at the ith stage) will be the index it has at the nth stage (which is just to say that event and begin points are never relocated). The bucket indices of event points and persistence begin points will never change; the index of a persistence endpoint will change at most once, if and when it is first clipped.

The buckets "accessible" at the ith stage (*i.e.* the indices into which points can be put at that stage) are limited to $2i$ (the current event bucket), $2i-1$ (the bucket for newly clipped persistence endpoints), and $2n+1$ (for unclipped persistence endpoints). So bucket positions established at previous stages cannot be changed (except for persistences that have been clipped). That is, at stage i the program cannot put points into, or take points out of, buckets prior to $2i-1$, thus cannot affect point-ordering relationships based on buckets with lower indices).

The ordering relations between pairs of points depend only on the points' respective bucket indices. Here are definitions for the point-ordering relations, assuming points p_1 and p_2 both are in $\texttt{point}_i$, and have bucket indices b_1 and b_2 respectively:

D.6.2 Definition of $\sim$

1. $(p_1, p_2) \in \sim_i$ if $p_1 = p_2$, or if $b_1 = b_2$ and b_1 and b_2 are even.

This can be verified directly in the code, function =p.

D.6.3 Definition of $\prec$

1. $(p_1, p_2) \in \prec_i$ just in case $b_1 < b_2$,

The function is $< p$.

D.6.4 Definition of $\preceq$

1. $(p_1, p_2) \in \preceq_i$ just in case $b_1 \leq b_2$.

The function is $=< p$.

It's easy to verify certain properties of the orderings so defined—transitivity of all three relations, that $\prec \Rightarrow \preceq$, that substituting $\sim$ points preserves $\sim$, $\prec$, and $\preceq$ relations, *etc*. We defer these proofs to appendix E.

E Proof that Program Models the Temporal Axioms

We prove here that the structure produced by the temporal program (represented by the relations $\mathtt{event}_n$, $\mathtt{persist}_n$, $\sim_n$, *etc.*) is a model of the axioms ($\mathbf{T} \cup \mathbf{P}$).

As far as the problem axioms ($\mathbf{P}$) go, we noted that every (*pcause*, *contradict*, and *event*) assertion in $\mathbf{P}$ is input to the program and is reflected unchanged in the output. Our restrictions also require that $\mathbf{P}$ have a set of $\prec$ assertions ordering the event points evt-pt_i. The order is reflected in the program's assumption that the events are input in temporal order; therefore by clause 2 of the definition of bucket and of $\prec$, $(\mathrm{PEVT}_i, \mathrm{PEVT}_j) \in \prec_n$ iff $1 \leq i < j \leq n$.

The axioms in $\mathbf{T}$, we noted, were implications (except for those concerning *ALWAYS*). We prove, by induction on i (event index), that all are true of the program's output. To do so we demonstrate that they're all true initially, then assume that they're true for the i^{th} stage and that all their antecedents hold at the $i{+}1^{st}$ stage, then we show that the consequents for all axioms must be true at the $i{+}1^{st}$ stage.

E.1 Point-ordering axioms

The point-ordering axioms (numbers 10–15) we can verify more directly. The ordering relations between any two points depends only on their respective bucket indices. As we saw in the definition of bucket, every element of point has exactly one index, which is an integer in the range $[0, \ldots, 2n + 1]$. We can thus use certain properties of the integers (*e.g.* transitivity) to verify the ordering axioms. The defining rules for these relations are in section D.6, on page 105. The point ordering axioms themselves begin on page 78 of

appendix A. We assume that there are individuals p_1, p_2, and p_3, all of which are members of $\texttt{point}_n$, and that they have bucket indices b_1, b_2, and b_2, respectively.

Axiom 10a ($\sim$ is reflexive): is true by definition of $\sim$.

Axiom 10b ($\sim$ is symmetric): is true by symmetry of equality over the integers.

Axiom 10c ($\sim$ is transitive): is true by transitivity of equality over the integers.

Axiom 11a ($\prec$ is non-reflexive): $b_1 \not< b_1$, so $(p_1, p_1) \notin \prec_n$

Axiom 11b ($\prec$ is antisymmetric): true by antisymmetry of $<$ over the integers.

Axiom 11c ($\prec$ is transitive): true by transitivity of $<$ over the integers.

Axiom 12a ($\preceq$ is transitive): true by transitivity of $\leq$ over the integers.

Axiom 12b (transitivity for $\prec$ and $\preceq$): true by transitivity of $<$ and $\leq$ over the integers.

Axiom 12c (transitivity for $\preceq$ and $\prec$): true by transitivity of $<$ and $\leq$ over the integers.

Axiom 13a ($\sim \Rightarrow \preceq$): because $b_1 = b_2 \Rightarrow b_1 \leq b_2$.

Axiom 13b ($\prec \Rightarrow \preceq$): because $b_1 < b_2 \Rightarrow b_1 \leq b_2$.

Axiom 14a ($\sim$ incompatible with converse $\prec$): if $(p_1, p_2) \in \sim_n$ then $b_1 = b_2$. Thus $b_2 \not< b_1$, and $(p_2, p_1) \notin \prec_n$

Axiom 14b ($\preceq$ incompatible with converse $\prec$): if $(p_1, p_2) \in \preceq_n$ then $b_1 \leq b_2$. Thus $b_2 \not< b_1$, and $(p_2, p_1) \notin \prec_n$

Axiom 15a–d (substituting $\sim$ points preserves $\prec$ and $\preceq$ relations): all four of these axioms follow from the fact that if $b_1 = b_2$, substituting b_1 for b_2 preserves relations $<$ and $\leq$ over the integers.

E.2 Clipping and point ordering—axioms 5–7

It's convenient to verify directly the validity of axioms relating persistences and clippings to relevant point orderings (axioms 5, 6, and 7).

Axiom 5 (clipping $\Rightarrow \prec$)[6]:

```
if  (1) (persist_i tok pat bp ep)
and (2) (clipped_i tok p2)
then    (≺_i ep p2)
```

Recall that the program reports that (tok, p_2) $\in$ clipped$_i$ just in case tok $\in$ persist$_i$, and p2 $\in$ point$_i$, and the bucket index of p_2, which is b_2, is greater than the bucket index of tok's end point, which is (b_1). But then from the definition of $\prec$ it immediately follows that ($\prec_i$ ep p_2).

Axiom 6 (unclipped persistences endure):

```
if  (1) (persist_i fact-tok fact-pat fact-bp fact-ep)
and (2) (point_i p)
and (3) (fact-tok, p) ∉ clipped_i
then    (≤_i p fact-ep)
```

[6]Note that for any relation R, we use "$(R_i\ \overline{x})$" to abbreviate "$\overline{x} \in R_i$".

Conversely, if `fact-tok` is a persistence at the i^{th} stage and p is a point at the i^{th} stage, the program will report that p does *not* clip `fact-tok` just in case the bucket of p is no greater than the bucket index of `fact-ep`. By definition, then, (p, `fact-ep`) $\in \preceq_i$.

Axiom 7 (begin point comes before end point):

```
if    (persist_i tok pat p1 p2)
then  (≺_i p1 p2)
```

Let $j < n$ be the stage at which tok was created, and b_1 and b_2 be the bucket indices of p_1 and p_2. Initially $b_1 = 2j$ and $b_2 = (2n + 1)$. Due to restrictions on the problem axioms, tok cannot be clipped at the j^{th} stage, so at the end of the j^{th} stage the inequality $b_1 < b_2$ holds, and $(p_1, p_2) \in \prec_j$. Tok's begin point, p_1, will stay in bucket $2j$ through all subsequent stages. If tok is never clipped, p_2 will stay in bucket $2n + 1$ as well, and the inequality holds thereafter. If tok is clipped, say at stage $k > j$, p_2's index becomes $(2k - 1)$ at that stage and thereafter. But since $(2k - 1) > 2j$ the inequality still holds. So it's always the case that $(b_1 < b_2)$, thus for every stage i, $(\prec_i p_1 p_2)$.

Now we proceed with the inductive proof for axioms 1–4 and 8. We start by proving the base case—that each axiom is true at stage 0.

E.3 Base case

Axiom 1 (creating a new persistence): Since $\mathbf{event}_0$ is empty, the precondition can never hold, and no persistences can be generated by this axiom at the 0^{th} stage.

Axiom 2a (ALWAYS token): The ALWAYS token, with begin and end points PBA and PEA, is explicitly in $\mathbf{persist}_0$.

Axiom 2b. The only points in $\mathbf{point}_0$ are PBA and PEA. The first instance of the axiom is satisfied, since (= PBA PBA), and since PBA is in bucket 0 and PEA is initially in bucket $2n + 1$, ($\prec_0$ PBA PEA) by definition.

Axiom 2c. Since $\mathbf{clipped}_0$ is empty, this axiom is true.

Axiom 3 (direct clipping): The ALWAYS token is the only one in $\mathbf{persist}_0$, and by restriction on P there can be no pattern contradictory to ALWAYS. So the antecedent for this axiom will not be true at the 0^{th} stage.

Axiom 4 (indirect clipping): $\mathbf{Clipped}_0$ is empty, so the antecedent can't be true.

Axiom 8a (persistence endpoints are points): True directly by definition of $\mathbf{persist}_0$ and $\mathbf{point}_0$.

Axiom 8b (event points are points): $\mathbf{Event}_0$ is empty, so the antecedent can't be true at the 0^{th} stage.

E.4 Induction step

We assume that all the axioms are true at the i^{th} stage, and go on to verify that they're true at the $i+1^{st}$.

Axiom 1:

```
if   (1) (pcause_{i+1} pc-tok ofact-pat evt-pat nfact-pat)
and  (2) (persist_{i+1} ofact-tok ofact-pat ofact-bp ofact-ep)
and  (3) (event_{i+1} evt-tok evt-pat PEVT)
and  (4) (≤_{i+1} ofact-bp PEVT)
and  (5) (≤_{i+1} PEVT ofact-ep)
then verify that
    (persist_{i+1}
      pt(pc-tok, evt-tok, ofact-tok)
```

```
      cfact-pat
      pb(pc-tok, evt-tok, ofact-tok)
      pe(pc-tok, evt-tok, ofact-tok))
and that
    (~i+1
      pb(pc-tok, evt-tok, ofact-tok)
      pe(pc-tok, evt-tok, ofact-tok))
```

As we saw in section 5.2 the proof involves an analysis of cases—for each of the five relations in the preconditions, we split on the case that the tuple was added at the i+1st stage, or was added at the i^{th} stage or before. While there are thus 32 possible cases to be verified for this example, we can eliminate most of them straight off. There's no point in splitting cases for clause (1), for example, because as far as the program is concerned, $pcause_0 = pcause_1 = \ldots = pcause_n$.

Next consider clauses (4) and (5): (4) involves a point-ordering relation over event and persistence begin points, and a look at the definition of bucket shows that once the bucket indices of event and begin points are fixed they are never changed, so the order of these points can't change from stage to stage either. As far as clause (5), if a persistence end point is ever ordered by the program to fall after an event point, it will *always* be ordered after that event point. (It could be clipped at some later stage, but only by a *subsequent* event.) So the *original* orderings imposed on the points in clauses (4) and (5) are all that matter, and we don't have to split those cases.

Finally, since we know exactly what event assertions are added at each stage, we know that if the particular event tuple named in clause (3) were added at the i+1ststage, then it must be the triple (tok-evt$_{i+1}$, pat-evt$_{i+1}$, pt-evt$_{i+1}$).

So it looks like we have to consider only these four cases:

1. both the **persist** and the **event** tuples (hereafter abbreviated by their token names, **ofact-tok** and **evt-tok**), were added at or before the i^{th}stage

2. **evt-tok** is the i^{th}event, or one previous, but **ofact-tok** was added at the $i+1^{st}$ stage

3. **ofact-tok** was added at the i^{th} stage or before, and **evt-tok** names the $i+1^{st}$ event

4. both the **ofact-tok** and **evt-tok** were added at the $i+1^{st}$stage

The first case is handled by the induction hypothesis. The second case is impossible—assume that **ofact-tok** is created at the $i+1^{st}$ stage and that **evt-tok** is the j^{th} event for $j \leq i$. Then ($\prec_{i+1}$ $pevt_j$ **ofact-bp**), and precondition (4) is thus impossible.

Next cases 3 and 4. If we assume preconditions (1) through (3) along with the restrictions of the third case (that **ofact-tok** was there at the i^{th} stage and **evt-tok** is the $i+1^{st}$ event), precondition (4) follows as well, because ($\prec_{i+1}$ **ofact-bp** $PEVT_{i+1}$). Case 4 satisfies this precondition too, because ($\sim_{i+1}$ **ofact-bp** $PEVT_{i+1}$) when **ofact-tok** comes into being at the $i+1^{st}$ stage. In both cases we have to verify that the appropriate persistence will be added at stage $i+1$. Looking at the definition of **persist** (page 100), our task reduces to showing that ($\preceq_{i+1}$ **ofact-bp pevt**) $\Rightarrow$ **ofact-tok** $\in$ **unclipped-toks**$_i$.

Assume to the contrary that **ofact-tok** $\notin$ **unclipped-toks**$_i$—in other words, it is clipped as of the i^{th} stage. There is therefore a stage $j \leq i$ at which **ofact-tok** is *first* clipped, and the bucket index of **ofact-ep** is $2j-1$. The bucket index of $PEVT_{i+1}$ is $2i+2$ in both case 3 and case 4, and $(2j-1) < (2i+2)$ since $j \leq i$. Therefore ($\prec_{i+1}$ **ofact-ep PEVT**), and it's impossible that ($\preceq_{i+1}$ **ofact-bp PEVT**).

Thankfully, proofs of the remaining axioms are somewhat less tedious:

Axioms 2a and b:

```
if   (1) (point_{i+1} p)
then  either
          (= p PBA)
or        (≺_{i+1} PBA p)


if   (1) (point_{i+1} p)
then      (PTA, p) ∉ clipped_{i+1}.
```

Recall that we're assuming the i^{th} stage true, so we're only interested in those points in $\texttt{point}_{i+1 \setminus i}$. All points added at that stage will be given either bucket index $2i$, or $2n+1$—in either case this is larger than 0, so the $\prec$ relation holds. For the second axiom, a token can be clipped at the $i+1^{st}$stage (item 4 in the definition of `clipped`) only if there's another persistence asserting a pattern contradictory to that token's. But we specifically prohibit any pattern be contradictory to `ALWAYS`, so the `PTA` token can't be clipped at the $i+1^{st}$ stage.

Axiom 3:

```
if   (1) (persist_{i+1} fact-tok fact-pat fact-bp fact-ep)
and  (2) (persist_{i+1} clip-tok clip-pat clip-bp clip-ep)
and  (3) (contradict_{i+1} fact-pat clip-pat)
and  (4) (≺_{i+1} fact-bp clip-bp)
then      (clipped_{i+1} fact-tok clip-bp)
```

As we did in verifying the persistence axiom, we can ignore splitting cases on `contradict` and $\prec$, and consider the following four:

1. both `fact-tok` and `clip-tok` existed at the i^{th} stage

2. `fact-tok` existed at the i^{th} stage, but `clip-tok` was created at the $i+1^{st}$ stage

3. clip-tok existed at the i^{th} stage, but fact-tok was created at the $i+1^{st}$ stage

4. both fact-tok and clip-tok were created at the $i+1^{st}$ stage.

Case 1 is covered by the induction hypothesis. Cases 3 and 4 fail on precondition (4): for case 3 we have ($\prec$ clip-bp fact-bp) and for case 4 we have ($\sim$ clip-bp fact-bp), so in either case precondition (4) cannot hold. We're left then with case 2 (and note that precondition (4) is satisfied).

For case 2, all antecedents are satisfied. It's either the case that fact-tok $\in$ unclipped-toks_{i+1}, or that fact-tok $\in$ $\text{clipped-toks}_{i+1}$. If the former holds then item 4 of clipped applies, and by that rule, (fact-tok, clip-bp) $\in$ clipped_{i+1}, and the consequent is true. Assume instead that fact-tok $\in$ $\text{clipped-toks}_{i+1}$, and that clip-bp $\in$ $\text{point}_{i+1 \setminus i}$. In this case item 3 of the definition for clipped is satisfied, and (fact-tok, clip-bp) $\in$ clipped_{i+1}. Again the axiom's consequent is satisfied.

Axiom 4:

```
if   (1) (persist_{i+1} fact-tok fact-pat fact-bp fact-ep)
     (2) (clipped_{i+1} fact-tok p_1)
     (3) (point_{i+1} p_2)
     (4) (≤_{i+1} p_1 p_2)
then     (clipped_{i+1} fact-tok p_2)
```

Let j be the stage at which fact-tok was created, let k be the stage at which p_1 first clips fact-tok, and let l be the stage at which p_2 becomes a point. If j, k, and l are all $\leq$ i, then the induction hypothesis applies. Consider, then, cases where at least one of them $= i + 1$. Note further that it must be that $k > j$ (because a fact can't be clipped before or at the same stage it's created), and $l \geq k$, because otherwise precondition (4) would fail. So that leaves us with two cases:

1. `fact-tok` created at stage $j < k$, clipped by p_1 at stage $k < i+1$, and p_2 becomes a point at stage $l = i+1$

2. `fact-tok` created at stage $j < k$, clipped by p_1 at stage $k = i+1$, and p_2 becomes a point at stage $l = i+1$.

In either case `fact-tok` $\in$ $\texttt{clipped-toks}_{i+1}$, and in either case $p_2 \in \texttt{point}_{i+1\setminus i}$. So in either case item 3 in the definition of `clipped` applies, and $(\texttt{fact-tok}, p_2) \in \texttt{clipped}_{i+1\setminus i}$.

Axiom 8:

```
if (1)   (persist_{i+1} tok pat p1 p2)
then     (point p1)

if (1)   (persist_{i+1} tok pat p1 p2)
then     (point p2)

if (1)   (event_{i+1}  tok pat p1)
then     (point p1)
```

In all three cases, if the (single) antecedent is true at the i^{th} stage then the induction hypothesis holds. If `tok` is a persistence created at the $i+1^{st}$ stage, or if `tok` names the $i+1^{st}$ event then the respective consequents are true by items 4 and 3 respectively, in the definition of `point`.

END

FILMED

2-86

DTIC