Technical Report No. 729

Department of
Computer Science

The Design of a Situation-Based Lygon Metainterpreter: I. Simple Changes and Persistence

Dr. G. White

Queen Mary and Westfield College

UNIVERSITY OF
LONDON

October 1996

729

The Design of a Situation-Based Lygon Metainterpreter: I. Simple Changes and Persistence.

Dr. G. White
Department of Computer Science
Queen Mary and Westfield College
Mile End Road, London
graham@dcs.qmw.ac.uk

October 9, 1996

... all things stedfastness doe hate
And changed be: yet being rightly wayd
They are not changed from their first estate;
But by their change their being doe dilate:
And turning to themselves at length again,
Doe work their own perfection so by fate;
Then over them Change doth not rule and raigne;
But they rayne over change, and doe their states maintaine.

Spenser

Contents

List of Tables

1 Introduction

The so-called "frame problem" is concerned with the computational representation of persistence and change, typically in situations which concern movements of, and changes in, everyday physical objects. There are extended uses of the term which apply to more general situations of persistence and change; I shall not be concerned with these.

A solution to the frame problem must, of course, fulfil several criteria, and there has been a good deal of progress lately in formulating such criteria in appropriately technical ways. [20, 21] However, we must not forget that these technical criteria are themselves answerable to less technical intuitions, and that these intuitions – non-technical though they may be – make up our basic understanding of the problem, from which our investigations start. In particular, the following two questions must always be asked:

- Do we understand the mathematics that we are using?
- Do we understand why the mathematics solves the problem that we are addressing?

The latter question, in particular, is an awkward one: it can be successfully answered for at least some areas of mathematics (calculus, for example, is the area of mathematics which deals with problems of continuity, and it is useful for solving problems which have to do with continuity; algebraic topology has to do with the relation between local and global, and is useful for problems in that area). And even answering the first question is a little tricky: answering it involves not merely an ability to reproduce proofs, but to explain why things are proved in the way they are, to understand why certain things are provable and certain things are not.

In this respect much current work on the frame problem does not do very well, even though it has its pages quite impressively adorned with complex formulae. We can distinguish two distinct approaches. One would be to say that the frame problem can be solved by "non-monotonic logic", without any further specification: but this is hardly enough to answer our questions. "Non-monotonic logic" is not a positive description of an area of logic: it simply denies that the logic possesses a certain characteristic. And so, not having a positive characterisation, we cannot give a very informative answer to the questions above: the reasons for using non-monotonic logic on the frame problem are simply the reasons for using logic-in-general on the frame problem, which are more or less the reasons for using logic-in-general on any problem at all. Now this may be a very wholesome thing to do, and much has been written on why it is a wholesome thing to do, but none of this (however inspirational it might be) does much explanation.

The other approach would be to specify some particular brand of non-monotonic logic, of which there are several. We can eliminate those that are based on epistemic intuitions, such as Reiter's default logic or Moore's autoepistemic logic; [5, Chapters 2 and 3] the frame problem can be posed in a completely non-epistemic setting, and seems to have, in that setting, all of its essential features.

Then there are various approaches based either on circumscription (i.e. minimising the extension of predicates by means of a suitable second order formula) or by using a preference relation on models. [5, Chapters 4 and 5] With all of this there are two basic problems. The first is that of accounting for the relevant concept – minimisation or the preference relation. What does it *mean*? What does it have to do with the frame problem? And the second is this: that such features (especially minimisation) behave extremely badly when combined with logical functors of mixed variance. For example, to minimise the extension of P is to maximise the extension of $\neg P$; and how are we to decide on whether P or $\neg P$ is the "real" predicate, which is to be minimised? The two problems interact here: because we do not understand what minimisation has to do with the frame problem, we find it difficult to give principled answers to the question of what should be minimised. And principled answers are not a luxury; they are a necessity when we have to deal with genuinely complex situations.

Then there are logics that have to do with classification and taxonomies, such as inheritance systems. [5, Chapter 8] Although these do seem to be principled ways of coping with problems of classification, their relevance for the frame problem seems limited; so far as we can understand it, it has nothing to do with taxonomy.

1.1 Linear Logic

We are proposing the use of linear logic to solve the frame problem. Linear logic is based on the idea of a *resource*: a resource is something which, after you've used it, you don't have it any more. This seems appropriate for the frame problem; if we consider, for example, the position of a block as a resource, then we can do nothing to it (in which case we still have it), or we can use it in an operation which replaces the position with a new position. So this seems at least to be the beginning of an answer to why linear logic should solve the frame problem.

The importance of linear logic for handling an analogous problem has been recognised for some while in the theoretical computer science community. There has, for a long while (since the 60's, at any rate) been a tendency to divide the behaviour of a machine into two parts, the *store* and the *environment*; the environment would consist of (λ-calculus like) variable bindings and the store would consist of rewritable memory locations. Whereas the environment enjoyed referential transparency and seemed to be amenable to standard mathematical techniques, the store emphatically could not be handled in a referentially transparent manner. There was, correspondingly, a tendency to regard it (and program concepts which used it, such as imperative assignment or objects) as somehow not susceptible to the usual techniques of mathematical analysis: as Abelson, Sussman and Sussman write, "no simple model with 'nice' mathematical properties can be an adequate framework for dealing with objects and assignment in programming languages". [1, p. 175]

However, when linear logic was discovered, it became evident that it might offer remarkable advantages in the analysis of mutable objects; by now a good deal of such analysis has been performed, with some success.

There has not been so much work on applications of linear logic to "real-world" problems (and, in particular, to the frame problem). However, there has been some work: as well as direct uses of linear logic, [15] there are two other approaches, which are each equivalent to linear proof search: [4, 12] the first uses Bibel's "connection method" with modifications to the proof search procedure which effectively turn it into a theorem prover for (multiplicative) linear logic. [10, 11] Secondly, there a formulation (again, essentially of multiplicative linear logic) as an equational theory using a single binary term, which corresponds to the linear tensor product. [13, 8]

Although this work has shown the computational viability of this approach, it is still somewhat limited in the fragment of linear logic that it can handle (it works entirely within the multiplicative fragment). We will extend the treatment to support a much larger fragment of linear logic, which will handle, for example, the Stolen Car Problem (for which one needs additive connectives). Furthermore, we can also, by using modal operators, handle ramification in a principled way (this will be described in a subsequent technical report): previous work on ramification using linear reasoning has been somewhat *ad hoc.* (Cf. [24])

1.2 Proof Theory and Reality

There is, however, one possible objection that we will have to deal with. It can be phrased as follows: that linear logic is naturally conceived in terms of proof-theory, and that, although various semantics for linear logic can be given, none of them is, on its own, completely satisfactory. Linear logic, although it can be *assigned* a semantics, is best conceived syntactically. And whereas a syntactically presented system may have good mathematical properties, unless it also has an intuitively meaningful semantics, its connection with reality is difficult to establish; as Barwise puts it,

> I am a grandstudent of Tarski. I grew up believing that semantics comes first, proof theory afterwards. If my career has been devoted to anything, it is to that belief. My current work, on semantic theories of information, is still in that direction. Trying to give an information-theoretic (in the semantic sense of information) foundation to logic and inference.
>
> So while I love the elegance of Gentzen systems, and did some work in them years ago, I genuinely have trouble understanding systems that are not given an intuitively meaningful semantics, whether formal or not, to motivate the rules of the system. [2]

One can respond to this argument as follows. Although it is tempting to align proof theory with some sort of purely mental construction, and similarly to say that semantics is how a theory becomes connected to reality, it is an alignment which is surprisingly difficult to sustain. Many "semantics" which are assigned to logical theories are themselves the result of Lindenbaum constructions, or term models, or the like – that is, they assign, to the terms of a theory, *mathematical constructions* rather than directly assigning elements of reality. It is (perhaps) true that correlations of the terms of a language with elements of reality will be a semantics in the mathematical sense (or, more precisely, correlations *of a certain sort*); however, this certainly does not establish that all mathematical semantics are correlations of the terms of a language with reality. So one half of the conventional position seems to fail: not all semantics gives us a connection with reality.

The other half of this position asserts that proof theory cannot be used to connect us with reality. It can be met by showing how one can view proof theory (at least in some cases) as establishing a connection between language and reality. This is, in the general case, a matter of some technical difficulty (a good deal of Girard's work on the geometry of interaction is concerned with establishing a presentation of proof theory in a sufficiently "coordinate-free" way for claims like this to make sense). However, in the simple sort of cases that we are considering, we can establish some connections.

A good place to start is with Davidson's work on actions. He is concerned to account for several linguistic phenomena, notably the fact that action sentences can have an arbitrary number of adverbial qualifiers (e.g. "Jones buttered the toast slowly, deliberately, in the bathroom, with a knife, at midnight" [7, pp. 106]). Davidson argues (on these grounds, and also on the grounds of other phenomena, such as anaphora), that a correct analysis should be something like

$$\exists x.\, \mathsf{buttered}(\mathsf{Jones},\ \mathsf{the\ toast}, x) \wedge \mathsf{slowly}(x) \wedge \ldots$$

where the variable x will stand for an *action* (of buttering the toast), which is identified by the first conjunct $\mathsf{buttered}(\mathsf{Jones},\ \mathsf{the\ toast}, x)$ and then further qualified by further conjuncts. Although Davidson does not identify what sort of component of reality such an x might be, he does argue fairly convincingly that there ought to be such things; there are many linguistic phenomena which seem to demand them for a proper analysis, and which also seem to show that such things would be components of reality rather than merely linguistic or mental entities.

What we should observe is that, if we represent the change from not buttered to buttered by a linear logic proof (and such a representation can in principle be constructed using the methods described in this report), then this proof can well fill in the role that x plays in the analysis above; such a proof, that is, will specify how the toast comes to be buttered, breaking the process down into constituent actions, and, from such an analysis as the proof performs, we can decide whether each of the adverbial predicates apply to this action or not. In other words, *actions are the referents of linear logic proofs*; this (if it can be established with careful enough arguments) will establish the sort of connection between linear logic and reality that we need. And it should be noted that *proofs* are precisely what we need here, because proofs are intensional; and, as is clear from the work of Davidson and others, actions are intensional. (See [28, IV§8]) When we describe an action, we describe, not merely the state transition that took place, but also how it was brought about; and it is precisely this sort of intensionality that linear proof theory gives us.

1.3 The Philosophy of State Transitions

Von Wright wrote a treatment of "the logic of action" [28] which has become the basis of much subsequent philosophical work. He first investigates the logic of state chance, using expressions like $p T q$ to denote the transition from a state where p obtains to a state where q obtains [28, Chapter II]; he then adds to this operators d and f, where $d(p T q)$ means that the agent in question *brings about* the state change from p to q. Similarly, $f(p\,T\,q)$ means that the agent refrains from bringing about the change from p to q.

This is an important and very early precursor of the linear logic approach, so it is worth discussing. On the whole, the formalism is remarkably similar – we would write $d(p\,T\,q)$ as $p \multimap q$ – with the following provisos:

- von Wright restricts himself to what could be called *decidable* state descriptions: that is, there is a set of atomic facts, and, for each such fact and for a given state vector, either p or its negation is in the state vector (where the negation here is not linear negation but some sort of classical negation operator: a similar formalism is used in [9]). We do not need this restriction, so our theory is somewhat more general.

- von Wright is working with classical modal logic, which means that, in our terms, he is working with only the additive connectives (except, of course, for $\cdot T \cdot$). Although a surprising amount can be done in this framework, it can become a little awkward, and uses decidability a good deal (see, for example, the surprisingly clumsy proof by enumeration in [28, II§8]).

- von Wright restricts his actions to cases where "the change ... does not happen, as we say, 'of itself', *i.e. independently of the action of the agent* ... If a door is so constructed that a spring or other mechanism pulls it open as soon as it has become closed, then there is no such *act* as the act of opening this door". [28, p. 43] This involves talk both of the agent's doings and of natural happenings in the world; although we could handle this, we will not do so at this stage.

- von Wright has a two-level theory, where one level consists of propositional descriptions of state changes and the other is generated by the modal operators $d(\cdot)$ and $f(\cdot)$ applied to state changes. We also have a two-level theory, but one level consists of *states*, not state transitions, and on the other level we have *proofs* of a successor state given an initial state and an action. We could introduce notation such as von Wright's $d(p\,T\,q)$ to stand for some sort of primitive action, of course, which would make a more direct translation possible: for example, equation (4), defines the action of loading, could equally well define it as $d(\,\mathsf{unloaded}\,T\,\,\mathsf{loaded})$.

2 Generalities

2.1 The Language

We have argued, then, that linear logic is a natural candidate for reasoning about the frame problem. We still have to decide how this logic is to be used computationally; we will use the linear logic programming language Lygon, [14] for several reasons.

The first is that logic programming is based on a *relational*, rather than a functional, paradigm. This is appropriate for our problem: we want to be able to tackle both temporal projection, reasoning forwards in time, and explanation, reasoning backwards, and even combinations of the two (the Stanford Murder Mystery problem[1] requires a combination of prediction and explanation). A functional language would give us a unidirectional solution. Furthermore, we want to be able to handle indeterminacy (for example, the Stolen Car Problem[2] does not have a determinate solution). Again, a functional approach would not handle this cleanly.

Secondly, it is clear from the above discussion that *proofs* are very significant: they model the intensionality of agency. So we might expect a Prolog-like language such as Lygon, in which execution can be regarded as proof search, to be appropriate; and this turns out to be the case.

Thirdly, Lygon allows the use of Prolog code (via the $\mathsf{prolog}(\cdot)$ predicate). This will be important for implementation, because we will be dealing with relatively large data structures (we want to be able to cope with model "worlds" of reasonable size, containing, say, a few thousand objects). Although it is theoretically possible to implement data structures directly in Lygon, it is computationally rather inefficient (it overloads the branch point stack), and calling a Prolog implementation turns out to be considerably better.

2.2 The Predicates

We are concerned, then, with developing a metainterpreter, for Lygon, which will implement something close to the situation calculus (it will not exactly be the situation calculus because the linear connectives are finer-grained than the classical connectives employed in the situation calculus). We will not use the characteristic situation calculus notation with a result function; instead, we will have a triadic relation, which we will write $\langle X \mid A \vdash Y\rangle$. Here X and Y can be thought of as state vectors; that is, they will behave like tensor products of atomic propositions. A will be, typically, a linear implication, and will bring

[1]See p. 25.
[2]See p. 28.

about a state change, erasing some propositions from the "input state", X, and putting others in their place in the "output state", Y. Given such a predicate, we can do a variety of entertaining and useful things; for example, we can chain state transitions together (by solving $\langle X \mid A \vdash Y\rangle \,\&\, \langle Y \mid A \vdash Z\rangle$), we can perform temporal projection (by chaining forwards from a known X) and we can perform explanation (by chaining backwards from a known Y).

This predicate will, as we have said, implement a metainterpreter for Lygon. The provability of $\langle X \mid A \vdash Y\rangle$ (maybe from premises Γ) will itself correspond to the validity of a sequent $\Gamma, X, A \vdash Y$. So we have two interpretations of $\langle X \mid A \vdash Y\rangle$; one in terms of state changes, and one in terms of linear provability. If we can show that these two interpretations agree with one another, this can be viewed as an expression of the principle that linear logic is a logic of state. In particular, linear logic proofs will represent state changes. Thus, if we have a situation with block $b1$ at position $l1$, then this can be represented by the proposition $\mathsf{at}(b1, l1)$, and the action of moving the block from $l1$ to $l2$ can be represented by the inference

$$\frac{\mathsf{at}(b1,l2) \vdash \mathsf{at}(b1,l2) \quad \mathsf{at}(b1,l1) \vdash \mathsf{at}(b1,l1)}{\mathsf{at}(b1,l1),\, \mathsf{at}(b1,l1) \multimap \mathsf{at}(b1,l2) \vdash \mathsf{at}(b1,l2)} \multimap \mathrm{L};$$

we should also notice that, if we have other, unchanged, elements of the state, these are handled correctly, as the inference in Table 1 shows.

However, besides wanting this metainterpreter to be logically correct, we also want it to be computationally efficient. In practice, this will mean the following. The state changes that are considered in AI are those in which one can have a large state vector, but in which, at any one time, only a small number of components of the state vector change: one may have, for example, a large number of objects in a room, but only a small number will change at a particular time. This seems to be realistic; as Shanahan says, "most fluents are unaffected by most actions". [22, p. 20] So we would like a metainterpreter which will do well in such situations.

This desideratum can be expressed in two ways: in terms of complexity, and in terms of proof. We are representing state changes by proofs: the identity state change ought to correspond to the identity proof. Consider a state change, represented by the sequent $\langle X \mid A \vdash Y\rangle$. Because most of the state vector is unaffected, we can write $X = X' \otimes Z$ and $Y = Y' \otimes Z$, where X' and Y' are the sub-vectors which play a role in the state change. "Restricted to Z", our proof must be the identity proof: i.e.

Desideratum 1. *We must have a proof* Π *of* $\langle X' \mid A \vdash Y'\rangle$ *, and our original proof must be of the form:*

$$\dfrac{\dfrac{\begin{array}{c}\Pi \\ \vdots \\ X', A \vdash Y'\end{array} \qquad \overline{Z \vdash Z}}{Z, X', A \vdash Z \otimes Y'} \otimes L}{Z \otimes X', A \vdash Z \otimes Y'} \otimes R \tag{1}$$

Notice that this is very similar to the deduction in Table 1, except that we have slightly enlarged our inference system to allow ourselves the identity

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{at}(b1,l2)\vdash\mathsf{at}(b1,l2)}\quad\dfrac{}{\mathsf{at}(b1,l1)\vdash\mathsf{at}(b1,l1)}}{\mathsf{at}(b1,l1),\ \mathsf{at}(b1,l1)\multimap\mathsf{at}(b1,l2)\vdash\mathsf{at}(b1,l2)}\multimap\mathrm{L}\qquad\dfrac{\dfrac{\dfrac{}{\mathsf{at}(b2,l3)\vdash\mathsf{at}(b2,l3)}\quad\overset{[1]}{\vdots}\ \mathsf{at}(b3,l4)\otimes\ldots\vdash\mathsf{at}(b3,l4)\otimes\ldots}{\mathsf{at}(b2,l3),\ \mathsf{at}(b3,l4)\otimes\ldots\vdash\mathsf{at}(b2,l3)\otimes\mathsf{at}(b3,l4)\otimes\ldots}\otimes\mathrm{R}}{\mathsf{at}(b2,l3)\otimes\mathsf{at}(b3,l4)\otimes\ldots\vdash\mathsf{at}(b2,l3)\otimes\mathsf{at}(b3,l4)\otimes\ldots}\otimes\mathrm{L}}{\mathsf{at}(b1,l1),\ \mathsf{at}(b2,l3)\otimes\mathsf{at}(b3,l4)\otimes\ldots,\ \mathsf{at}(b1,l1)\multimap\mathsf{at}(b1,l2)\vdash\mathsf{at}(b1,l2)\otimes\mathsf{at}(b2,l3)\otimes\mathsf{at}(b3,l4)\otimes\ldots}}{\mathsf{at}(b1,l1)\otimes\mathsf{at}(b2,l3)\otimes\mathsf{at}(b3,l4)\otimes\ldots,\ \mathsf{at}(b1,l1)\multimap\mathsf{at}(b1,l2)\vdash\mathsf{at}(b1,l2)\otimes\mathsf{at}(b2,l3)\otimes\mathsf{at}(b3,l4)\otimes\ldots}\otimes\mathrm{L}
$$

Table 1: A Linear Inference Representing a State Change

sequent on the (possibly large) state vector Z without having to resolve it into components.

We can reformulate this using propositional quantification:

Desideratum 2. *Our proof must arise from a proof of* $\vdash \Lambda Z.\ Z \otimes X', A \multimap Z \otimes Y'$ *by substitution for the propositional variable* Z *and a subsequent application of* $\multimap R$.

This leads to the complexity-theoretic form of our desideratum:

Desideratum 3. *Whatever the complexity of the search for the subproof* Π *might be, the search for the "Z-component" ought to take constant time: that is, the size of* Z *ought to play no role in the complexity of proof search.*

3 Logical Form

Lygon is based on the idea of uniform proof. This can be described as follows: we define recursively two classes of formulae – the definite formulae, $\mathcal{D}$, and the goal formulae, $\mathcal{G}$ – and we consider sequents of the form $\mathcal{D} \vdash \mathcal{G}$, where $\mathcal{D}$ is here a multiset of definite formulae and $\mathcal{G}$ a multiset of goal formulae. For suitable choices of $\mathcal{D}$ and $\mathcal{G}$, one can then show that such sequents can always be proved in a *goal-directed* way; that is, that (with the exception of the so-called *resolution* rules) we can always use sequent calculus rules whose principal connectives are in $\mathcal{G}$. The resolution steps – even though they have principal connectives in $\mathcal{D}$ – can be selected by the choice of a formula in $\mathcal{G}$. So, in practice, proof search in Lygon can be governed by examining the formulae in $\mathcal{G}$: that is, it is *goal-directed.* In typical situations this is computationally efficient: that is, $\mathcal{D}$ is large and fixed, and can thus be considered as a program, whereas $\mathcal{G}$ is small and not fixed, and can thus be considered as input data.

We should remark here that, because of the duality properties of linear logic, the fact that we generally write such sequents with $\mathcal{G}$ on the right and $\mathcal{D}$ on the left is merely conventional; proofs of a sequent with, let us say, a formula A on the right are trivially equivalent to proofs of a sequent with $A^{\perp}$ on the left. So the theory of uniform proof would also apply to sequents of the form $\vdash \mathcal{D}^{\perp}, \mathcal{G}$, or, for that matter, with some mixture of $\mathcal{G}$ and $\mathcal{D}^{\perp}$ on the right and some mixture of $\mathcal{D}$ and $\mathcal{G}^{\perp}$ on the left. All that matters is that, on each side, we can divide the formulae into active and passive in such a way that we can always prove such sequents using a proof search strategy which is directed by the active formulae.

Given, then, our predicate $\langle X \mid A \vdash Y \rangle$, which of its arguments should be active and which should be passive? We know that X and Y are typically large, whereas A is typically small; A, then, should be active, whereas X and Y should be passive. This is because uniform proof proceeds by selecting principal connectives in goal formulae; if the goal formulae are small, then this search will be rapid. At the end of the proof, of course, the proof has to resolve the atomic components of the goal formulae with the definite formulae; however, if X and Y are stored in some appropriately indexed data type, then this resolution can be done efficiently, despite the size of X and Y.

This gives us some restrictions on the logical form of X and Y. We are looking, let us remember, at sequents of the form $\Gamma, A, X \vdash Y$. For standard

Lygon, $\mathcal{D}$ and $\mathcal{G}$ are defined as follows, where α stands for the class of atomic formulae. (see [17, Definition 2.23]):

$$
\begin{array}{rcl}
\mathcal{D} & \longrightarrow & \alpha \mid \mathbf{1} \mid \bot \mid \mathcal{D}\otimes\mathcal{D} \mid \mathcal{D}⅋\mathcal{D} \mid \mathcal{D}\&\mathcal{D} \mid \\
& & \mathcal{G}\multimap\alpha \mid \mathcal{G}\multimap\bot \mid \bigwedge_x.\mathcal{D} \mid !\mathcal{D} \\
\mathcal{G} & \longrightarrow & \alpha \mid \mathbf{1} \mid \bot \mid \top \mid \mathcal{G}\otimes\mathcal{G} \mid \mathcal{G}⅋\mathcal{G} \mid \mathcal{G}\&\mathcal{G} \mid \mathcal{G}\oplus\mathcal{G} \mid \\
& & \mathcal{D}\multimap\mathcal{G} \mid \bigwedge_x.\mathcal{G} \mid \bigvee_x.\mathcal{G} \mid !\mathcal{G} \mid ?\mathcal{G}
\end{array}
$$

These are, of course, definitions for $\mathcal{D}$ on the left of a sequent and $\mathcal{G}$ on the right; we may derive from these definitions of the dual classes, i.e. $\mathcal{G}^\perp$ – goal formulae on the left of a sequent – and $\mathcal{D}^\perp$ – definite formulae on the right.

In the sequent corresponding to $\langle X \mid A \vdash Y\rangle$, X should be a definite formula, and is on the left of the sequent: that is, X should be in $\mathcal{D}$. On the other hand, Y is a definite formula on the right, so we have $Y \in \mathcal{D}^\perp$. However, we want to be able to chain these sequents (by writing something like $\langle X \mid A \vdash Y\rangle \mathbin{\&} \langle Y \mid B \vdash Z\rangle$), so that the same formulae ought to be able to occur in the X-position and in the Y-position. So X and Y ought certainly to be in $\mathcal{D}\cap\mathcal{D}^\perp$.

However, there are further restrictions. The formulae $A\otimes(B⅋C)$ and $(A\otimes B)⅋C$ both belong to $\mathcal{D}\cap\mathcal{D}^\perp$; however, we have the valid inference

$$
\dfrac{\dfrac{\dfrac{A\vdash A \qquad \dfrac{B\vdash C \quad B\vdash C}{B⅋C\vdash B, C}\,⅋\text{L}}{A,(B⅋C)\vdash(A\otimes B),C}\,\otimes\text{R}}{A\otimes(B⅋C)\vdash(A\otimes B),C}\,\otimes\text{L}}{A\otimes(B⅋C)\vdash(A\otimes B)⅋C}\,⅋\text{R} \tag{2}
$$

So, if we allowed passive formulae to be of this form, we would have the valid state change $\langle A\otimes(B⅋C) \mid \mathbf{1} \vdash (A\otimes B)⅋C\rangle$ without any active formulae at all. This is problematic, for several reasons.

The first is that it is inefficient computationally. Goal-directed proof search would handle such inferences by searching for them whenever the active formula context became empty. (Cf. the rule for an empty context given in [17].) If the passive formulae were large, this could impose an undue penalty. This can be thought of as a reformulation of the identity desideratum: the desideratum would say that the sequent $\langle X \mid \mathbf{1} \vdash X'\rangle$ should only have (at most) one solution, corresponding to the identity proof with $X = X'$.

The second is that, from the point of view of phenomena we are modelling (that is, state changes against a background of mostly unchanging objects), such inferences would correspond to nondeterministic spontaneous changes in the background. Although they do genuinely happen (bookshelves fall over without provocation, things go bump in the night, and so on), such happenings tend to be rather deliberately ruled out as part of the idealisation implicit in the frame problem. Although there has been some work on spontaneous evolution of the background (Reiter calls such evolution steps "natural actions" [18]), it is definitely an extension of the frame problem as originally posed. Such work is also still rather experimental: indeed, Reiter's treatment only covers natural actions which "must occur at their predetermined times", rather than the nonderministic spontaneous actions which inferences such as (2) would admit. At least as a first approximation, then, we are probably justified in ignoring such changes.[3]

If we wish to rule out such behaviour, then, we will have to restrict the syntax of passive formulae: a little thought reveals that they will have to be of the form $\bigotimes_{i \in I} A_i$, where the A_i are atoms.

The active formula A, of course, should come from the class $\mathcal{G}^{\perp}$; there is no need to write down a definition of this.

4 Rules

We are now in a position to write down some rules: see Table 2. As we have seen, proof-theoretic treatments of logic programming have placed the goal formulae $\mathcal{G}$ on the right of sequents, and the definite formulae $\mathcal{D}$ on the left. This is not obligatory: the basic difference between $\mathcal{G}$ and $\mathcal{D}$ is that the former are "active" – their principal connectives are selected as the proof is constructed – whereas the latter are "passive" – their role in the proof is limited to the resolution and tensor steps, and those steps are actuated by selecting active formulae. Consequently, if we dualise the rules appropriately, we can allow both active and passive formulae on either side.

Because of its intended application to the situation calculus, our original notation – $\langle X \mid A \vdash Y \rangle$ – had passive formulae (X and Y) on both sides, but an active formula (A) only on the left. However, active formulae on the right may well be introduced in the course of a proof, so that we will have to change our notation. If we were being pedantic, we could write our sequents with two regions on each side (something like $\langle X \mid A \vdash B \mid Y \rangle$), but this would be unnecessarily cumbersome; what we really want to keep track of is the fate of the state vectors X and Y, so we shall simply write our sequents $\Gamma\ \langle X \vdash Y \rangle\ \Delta$, where X and Y are state vectors and where Γ and Δ are the other formulae in the context. X and Y are, of course, passive; there may be other passive formulae around (typically exponentiated components of Γ which represent "laws of nature"), but we have simply assumed, when writing down the rules, that they make sense, i.e. that the non-resolution rules always have active principal formulae. In practice, this does not cause any significant ambiguity. Similarly, we will also ignore questions of permutability with passive formulae that are not represented in $\langle X \vdash Y \rangle$; these formulae will be handled as usual by Lygon's proof search mechanism, and the proofs of permutability will go through for them without change.

The single premise rules, and the environment copying rules, present little difficulty. They are clearly sound – that is, if the premises are valid sequents (with the intended interpretation of $\langle \cdot \vdash \cdot \rangle$), then so is the conclusion. For completeness, we need to establish the following:

Proposition 1. *The single premise rules and the environment copying rules*

[3] And it may be that we can cite Aristotle in support: Τοῦ δ' ἀπὸ τύχης οὐκ ἔστιν ἐπιστήμη δι' ἀποδείξεως. οὔτε γὰρ ὡς ἀναγκαῖον οὔθ' ὡς ἐπὶ τὸ πολὺ τὸ ἀπὸ τύχης ἐστιν, ἀλλὰ τὸ παρὰ ταῦτα γινόμενον· ἡ δ' ἀπόδειχις θατέρου τούτων. (*Anal. Post.* 87^b19f.) However, the interpretation of Aristotle's doctrine of chance events is notoriously controversial – see [23] – so one should probably beware of glib citation.

Single Premise Rules:

$$\frac{\Gamma,\ \langle X \vdash Y\rangle\ \delta_1, \delta_2, \Delta}{\Gamma,\ \langle X \vdash Y\rangle\ \delta_1 ⅋ \delta_2, \Delta}\ ⅋\,\mathrm{R} \qquad \frac{\Gamma, \gamma_1, \gamma_2\ \langle X \vdash Y\rangle\ \Delta}{\Gamma, \gamma_1 \otimes \gamma_2\ \langle X \vdash Y\rangle\ \Delta}\ \otimes\mathrm{L}$$

$$\frac{\Gamma\ \langle X \vdash Y\rangle\ \delta_1, \Delta}{\Gamma\ \langle X \vdash Y\rangle\ \delta_1 \oplus \delta_2, \Delta}\ \oplus\mathrm{R1} \qquad \frac{\Gamma\ \langle X \vdash Y\rangle\ \delta_2, \Delta}{\Gamma\ \langle X \vdash Y\rangle\ \delta_1 \oplus \delta_2, \Delta}\ \oplus\mathrm{R2}$$

$$\frac{\Gamma, \gamma_1\ \langle X \vdash Y\rangle\ \Delta}{\Gamma, \gamma_1 \& \gamma_2\ \langle X \vdash Y\rangle\ \Delta}\ \&\,\mathrm{L1} \qquad \frac{\Gamma, \gamma_2\ \langle X \vdash Y\rangle\ \Delta}{\Gamma, \gamma_1 \& \gamma_2\ \langle X \vdash Y\rangle\ \Delta}\ \&\,\mathrm{L2}$$

$$\frac{\Gamma, \delta\ \langle X \vdash Y\rangle\ \Delta}{\Gamma\ \langle X \vdash Y\rangle\ \delta^{\perp}, \Delta}\ {}^{\perp}\mathrm{R} \qquad \frac{\Gamma\ \langle X \vdash Y\rangle\ \gamma, \Delta}{\Gamma, \gamma^{\perp}\ \langle X \vdash Y\rangle\ \Delta}\ {}^{\perp}\mathrm{L}$$

Environment Copying Rules:

$$\frac{\Gamma\ \langle X \vdash Y\rangle\ \delta_1, \Delta \quad \Gamma\ \langle X \vdash Y\rangle\ \delta_2, \Delta}{\Gamma\ \langle X \vdash Y\rangle\ \delta_1 \& \delta_2, \Delta}\ \&\,\mathrm{R}$$

$$\frac{\Gamma, \gamma_1\ \langle X \vdash Y\rangle\ \Delta \quad \Gamma, \gamma_2\ \langle X \vdash Y\rangle\ \Delta}{\Gamma, \gamma_1 \oplus \gamma_2\ \langle X \vdash Y\rangle\ \Delta}\ \oplus\mathrm{L}$$

Environment Splitting Rules:

$$\frac{\Gamma_1\ \langle X_1 \vdash Y_1\rangle\ \delta_1, \Delta_1 \quad \Gamma_2\ \langle X_2 \vdash Y_2\rangle\ \delta_2, \Delta_2}{\Gamma_1, \Gamma_2\ \langle X_1 \otimes X_2 \vdash Y_1 ⅋ Y_2\rangle\ \delta_1 \otimes \delta_2, \Delta_1, \Delta_2}\ \otimes\mathrm{R}$$

$$\frac{\Gamma_1, \gamma_1\ \langle X_1 \vdash Y_1\rangle\ \Delta_1 \quad \Gamma_2, \gamma_2\ \langle X_2 \vdash Y_2\rangle\ \Delta_2}{\Gamma_1, \Gamma_2, \gamma_1 ⅋ \gamma_2\ \langle X_1 \otimes X_2 \vdash Y_1 ⅋ Y_2\rangle\ \Delta_1, \Delta_2}\ ⅋\,\mathrm{L}$$

Resolution Rules:

$$\frac{\Gamma_1\ \langle X \vdash Y\rangle\ \Delta_2 \quad \overline{!\Gamma_2, y\ \langle \mathbf{1} \vdash y\rangle\ ?\Delta_2}}{\Gamma_1, !\Gamma_2, y\ \langle X \vdash Y \otimes y\rangle\ \Delta_1, ?\Delta_2}\ \mathrm{resL} \qquad \frac{}{!\Gamma\ \langle x \vdash \perp\rangle\ x, ?\Delta}\ \mathrm{resR}$$

$$\frac{}{!\Gamma, x\ \langle \mathbf{1} \vdash \perp\rangle\ x, ?\Delta}\ \mathrm{Ax} \qquad \frac{}{!\Gamma\ \langle X \vdash X\rangle\ ?\Delta}\ \mathrm{Eq}$$

Here the spaces of the predicate $\langle \cdot \vdash \cdot\rangle$ can be occupied either by a state vector or by $\perp$; $⅋$ of two state vectors is undefined, and $Y_1 ⅋ Y_2$ is only defined when at least one of the Y's is a $\perp$.

Table 2: Situation Calculus Sequent Rules

given are sufficient for proof search: i.e. if there is a proof whose lowest active formula rule is a single premise rule or an environment copying rule, and which has one or more passive formula rules below that rule, then the passive formula rules can all be permuted above the given rule.

Proof The only passive formula rules that we are concerned with are $\otimes$ L and $\otimes$ R; by the table of permutabilities [25, p. 340] this can always be done. □

4.1 Environment-Splitting Rules

The environment-splitting rules, however, take a little more consideration. We will first review what happens in the one-sided case.

4.1.1 The One-Sided Case

In the one-sided case, goal formulae only occur on the right, and we solely have to consider right rules; environment splitting, then, only occurs with the rule $\otimes$ R. This rule takes the following form (see [17, p. 192]): we suppose that our sequent can be written

$$\mathcal{D}, C \vdash G_1 \otimes G_2, \mathcal{G} \tag{3}$$

where the left hand side is a multiset of clauses – these will be clausal decompositions of definite formulae – including the clause C, and the right hand side is a multiset of goal formulae. The application of the tensor rule will come in two stages: not all the left rules are permutable with the right tensor rule, so we must be able to apply a certain number of left rules before we can get round to $\otimes$ R. We can assume that the formulae in $\mathcal{D}$ have already been decomposed to some extent (for example, that any tensor products occurring at the top level have been expanded); this can be thought of as a sort of "compile-time" expansion of the definite formulae. The products of this expansion are called *clauses.* However, the necessary expansion cannot all be done at compile time; the exponentials, for instance, can give rise to an unbounded number of expanded formulae, simply by applying the contraction rule an arbitrary number of times. Furthermore, if we have a formula in $\mathcal{D}$ whose principal connective is $\invamp$, it cannot be split at compile time, since splitting it involves dividing the environment. So some expansion takes place during proof search, at "run-time"; Pym and Harland call the products of this expansion *components.* For example, consider the entailment

$$(A \otimes B) \invamp C \vdash (B \otimes A) \invamp C$$

The application of the $\invamp$ R rule to the goal gives

$$(A \otimes B) \invamp C \vdash (B \otimes A), C$$

We now have to apply the $\invamp$ L rule to the definite formula: this decomposes the sequent into the components

$$A, B \vdash B \otimes A \quad C \vdash C$$

and from here one merely needs to apply $\otimes$ R.

The general case of the $\otimes$R rule is very similar. If we are faced with an entailment of the form (3), we first decompose the left hand side into clauses and then the entailment into components, until we arrive at something of the form:

$$\mathcal{D}_0, \mathcal{D}'_0 \vdash G_1 \otimes G_2, \mathcal{G}_0, \mathcal{G}'_0 \quad \mathcal{D}_1 \vdash \mathcal{G}_1 \quad \ldots \quad \mathcal{D}_n \vdash \mathcal{G}_n$$

we then apply $\otimes$R to the first sequent, obtaining

$$\mathcal{D}_0 \vdash G_1, \mathcal{G}_0 \quad \mathcal{D}'_0 \vdash G_2, \mathcal{G}'_0 \quad \mathcal{D}_1 \vdash \mathcal{G}_1 \quad \ldots \quad \mathcal{D}_n \vdash \mathcal{G}_n$$

4.1.2 The Two-sided Case

By contrast, our case is both simpler and more complex. It is (trivially) more complex because we want passive and active formulae on both sides. This means that there are now two environment-splitting rules, and the same considerations apply to both. However, it is also substantially simpler, because the only definite formulae that we have to handle explicitly are the state vectors, which are much simpler than definite formulae in general. (There might be definite formulae in the environment which are not state vectors, but we can leave it up to Lygon's usual mechanisms to handle these).

We have, then,

Proposition 2. *By permuting inferences, applications of $\otimes$ R can be put into the form*

$$\frac{\Gamma_1 \langle X_1 \vdash Y \rangle \Delta_1 \quad \Gamma_2 \langle X_2 \vdash \perp \rangle \Delta_2}{\Gamma_1, \Gamma_2 \langle X_1 \otimes X_2 \vdash Y \rangle \Delta_1, \Delta_2} \otimes R$$

or into the form

$$\frac{\Gamma_1 \langle X_1 \vdash \perp \rangle \Delta_1 \quad \Gamma_2 \langle X_2 \vdash Y \rangle \Delta_2}{\Gamma_1, \Gamma_2 \langle X_1 \otimes X_2 \vdash Y \rangle \Delta_1, \Delta_2} \otimes R,$$

and similarly for applications of $⅋$ L.

Proof We have to examine which passive formula rules are impermutable with $\otimes$R (or, respectively, $⅋$L). Our passive formulae only have $\otimes$ as a connective, so the only passive formula rules that we have to consider are $\otimes$R and $\otimes$L. $\otimes$R can be permuted above anything, whereas $\otimes$L cannot, in general, be permuted above environment splitting rules. ([25, p. 340], *mutatis mutandis.*) After permuting all of the other passive rules above our application of $\otimes$R to an active formula, we are left with $\otimes$L below $\otimes$R: this gives inferences of the above form. □

Notice that none of this requires us to decompose the entailment into components; this is because there are no passive environment-splitting rules that are impermutable with active environment-splitting rules.

Normally, Lygon would this system by decomposing the left-hand state vector into its atoms at the start of the deduction. However, this would be unacceptably cumbersome due to the size of the state vectors; we do have to apply $\otimes$L to the the left vector, but we want to do so as little as possible. We have to apply it once before each environment-splitting rule, but that is all.

5 Two-Sided Resolution

We have now shown that we can transform any proof of one of our sequents into a form where (apart from the stated exceptions) the passive formula rules all occur above the active formula rules. We must, then, consider how to deal with these rules. We can assume that the active formula rules have been applied as far as possible, i.e. that we have a sequent of the form $\Gamma \langle X \vdash Y \rangle \Delta$, where Γ and Δ consist either of atoms or of passive formulae other than state vectors. Furthermore, we can assume that Γ does not contain $\mathbf{1}$ or $\mathbf{0}$, and that Δ does not contain $\top$ or $\bot$ (otherwise the appropriate active formula rules could have been applied).

First we need:

Lemma 1. *If $!\Gamma_0, \Gamma \langle X \vdash Y \rangle \Delta$ is provable by means of a sequence of resolution rules, where Γ and Δ consist of linear atoms, then it must be one of the following:*

- $!\Gamma_0 \langle \mathbf{1} \vdash \mathbf{1} \rangle$,
- $!\Gamma_0, A \langle \mathbf{1} \vdash \bot \rangle A$,
- $!\Gamma_0, A \langle \mathbf{1} \vdash A \rangle$
- $!\Gamma_0, A_1, \ldots A_i \langle A_{i+1} \otimes \ldots \otimes A_n \vdash A_1 \otimes \ldots \otimes A_n \rangle$

Proof A trivial induction. □

We can now establish:

Proposition 3. *If $\Gamma \langle X \vdash Y \rangle \Delta$ is a provable sequent where Γ and Δ are either atoms or are passive, then it has a proof which uses only the resolution rules.*

Proof We proceed by cases, examining the rule applied in the bottom step of the proof.

Axiom The sequent must be of the form $!\Gamma_0, A \vdash A$, with A an atom. There are four subcases:

- $\Gamma = !\Gamma_0 \cup \{A\}$, $\Delta = \{A\}$.
- $\Gamma = !\Gamma_0 \cup \{A\}$, $Y = A$. We apply resL, followed by Eq to close the left hand branch.
- $X = A$, $\Delta = \{A\}$. We apply resR.
- $X = Y = A$. We apply Eq.

$\otimes$ R or $\otimes$ L We can conveniently treat these rules together. We may have several applications of $\otimes$ L with an application of $\otimes$ R above it; after some manipulation, we can arrive at something of the form

$$\frac{\begin{array}{c}\Pi_1 \\ \vdots \\ !\Gamma_0, \Gamma_1 \langle X_1 \vdash Y_1 \rangle \Delta_1\end{array} \qquad \begin{array}{c}\Pi_2 \\ \vdots \\ !\Gamma_0, \Gamma_2 \langle X_2 \vdash Y_2 \rangle \Delta_2\end{array}}{!\Gamma_0, \Gamma_1, \Gamma_2 \langle X_1 \otimes X_2 \vdash Y_1 \otimes Y_2 \rangle \Delta_1, \Delta_2}$$

where Π_1 and Π_2 can inductively be assumed to be of the desired form. Now we use the lemma to establish the possible forms for the two sequents $!\Gamma_0, \Gamma_1 \langle X_1 \vdash Y_1 \rangle \Delta_1$ and $!\Gamma_0, \Gamma_2 \langle X_2 \vdash Y_2 \rangle \Delta_2$; an argument by cases shows that, for each feasible combination, the original sequent can be proved using resolution rules.

□

We have, then, a version of resolution proof for our restricted form of two-sided sequents, and we have seen that it is sound and complete. We can now worry about implementation.

6 Implementation

In an implementation, we will represent these different collections of formulae as follows. We will have a two-place Lygon predicate, which we will write $\mathsf{seq}(X, Y)$, where X and Y are state vectors: this predicate will represent our $\langle X \vdash Y\rangle$. It will be executed in an environment containing the elements of Γ and Δ; the active members of these will be encoded in the form $\mathsf{left}(\gamma)$ and $\mathsf{right}(\delta)$, respectively. Since it is the elements of Γ and Δ which drive the (goal-directed) proof-search, this can be simulated in our metainterpreter by giving suitable clauses for terms in their representations, clauses which match their principal connectives.

Consider, for example, the term $\mathsf{right}(A \parr B)$. This term will match the rule

$$\mathsf{right}(A \parr B) \quad \leftarrow \quad \mathsf{right}(A) \parr \mathsf{right}(B);$$

the term will thus be replaced by a $\parr$ of terms in the same environment. Now when we are verifying the correctness of a rule like this, there are two things to check:

1. that the environments are correct: that is, that each environment has exactly one clause of the form $\langle X \vdash Y\rangle$ in it, and that there are the right number of environments;
2. that the distribution of resources among the environments is correct.

In the case of this rule, these conditions are both easy to verify; if the environment was correct beforehand (that is, if it had exactly one $\mathsf{seq}(X, Y)$, and if it had the correct resources), then it will be correct afterwards, since it remains a single environment with exactly the same resources, and, according to the rule $\parr$ R, this is how it ought to behave. The same analysis holds for all of the single premise rules: that is, for $\parr$ R, $\otimes$ L, $\oplus$ R1, $\oplus$ R2, & L1, & L2, $^{\perp}$ L and $^{\perp}$ R.

Consider now the connective & on the right. The corresponding clause will be

$$\mathsf{right}(A \& B) \quad \leftarrow \quad \mathsf{right}(A) \& \mathsf{right}(B).$$

On execution (and subsequent execution of the external &) the original context will be replaced by two contexts, with exactly the same resources, except for the replacement of $A\&B$ by A in one and B in the other. So both criteria are satisfied: the environment has been correctly replaced by two environments, both of them have only one clause of the form $\langle X \vdash Y\rangle$ in them, and both of them have exactly the same resources as before (apart from the replacement of $A\&B$). The same goes for the clause corresponding to $\oplus$ on the left; this accounts for the rules in the second group.

The problem, however, comes with the rules $\otimes$ R and $\parr$ L; that is, the rules which split environments. The first problem that we shall face is this: what form do the sequent calculus rules take for our two-sided calculus? Having decided this, we will have to decide how to implement those rules in our metainterpreter.

The answer to the first question is clear. An environment-splitting rule such as

$$\frac{\Gamma_1\ \langle X_1 \vdash Y_1\rangle\ \delta_1, \Delta_1 \quad \Gamma_2\ \langle X_2 \vdash Y_2\rangle\ \delta_2, \Delta_2}{\Gamma_1, \Gamma_2\ \langle X_1 \otimes X_2 \vdash Y_1 \parr Y_2\rangle\ \delta_1 \otimes \delta_2, \Delta_1, \Delta_2} \otimes \mathrm{R}$$

can be simulated by a Lygon clause

$$\begin{aligned}(\mathsf{seq}(X_1 \otimes X_2, Y_1 \parr Y_2) \parr \mathsf{right}(A \otimes B)) \quad \leftarrow \quad & (\mathsf{seq}(X_1, Y_1) \parr \mathsf{right}(A)) \otimes \\ & (\mathsf{seq}(X_2, Y_2) \parr \mathsf{right}(B)).\end{aligned}$$

If there are other propositions in the environment, then they will be split correctly by Lygon's handling of the top-level $\otimes$s in the above rule and others like it.

However, there is one thing we have ignored; that is, the splitting of the state formulae X and Y.

6.1 Realisability

Suppose, then, that we consider the initial X and Y as global objects. As we have said, they will in practice be large tensor products of atoms, and they can thus be regarded as large vectors. Correspondingly, let such an atom be called a *component* of the corresponding vector. Consider one of the components of one of these vectors – a component of X, let us say. If we follow it up from the root of the proof tree, it will go one way or the other at the environment splitting rules, it will be duplicated at the environment copying rules, and otherwise it will be unaffected. It will thus correspond to a subtree of the proof tree, which forks only at the environment copying rules. Similarly, the whole vector Y will go one way or another at the environment splitting rules and the environment copying rules, and will be split up at the left resolution rules. We thus have a pair of functions, from the components of X (respectively Y) to the set of subtrees of the proof tree. These functions, of course, must satisfy certain conditions, namely those in Table 3. The advantage of this viewpoint is that X and Y (together with the trajectories of their components) are now global objects: we do not have to make any decisions about splitting X or Y when we apply active rules – what we rather have to do is to make appropriate decisions about the fate of the components of X and Y when we apply resolution and axiom rules, and, using these decisions, we can synthesise trajectories for all of the components of X and Y. The evaluation of the constraints on such decisions, and the synthesis of the resulting functions, can be carried out, lazily and deterministically, using global variables in Lygon.

Table 3 uses some notation, which is defined as follows. We label sequents (i.e. nodes of the proof tree) by tags, which are defined thus:

Definition 1. A *tag* is a string over the alphabet $\{l, r\}$. The empty string is written $[]$, and singleton strings are written $[l]$ and $[r]$. If τ_1 and τ_2 are tags, their concatenation is written $\tau_1 \cdot \tau_2$. Tags such as $[t_1] \cdot [t_2] \cdot [t_3] \ldots$ are written $[t_1, t_2, t_3 \ldots]$. By abuse of notation, $\tau \cdot [t]$ is written $\tau \cdot t$.

Tags are attached to sequents according to the following rules:

Single Premise Rules:

$$\frac{\Gamma,\ \langle f \vdash_\tau g\rangle\ \delta_1, \delta_2, \Delta}{\Gamma,\ \langle f \vdash_\tau g\rangle\ \delta_1 \parr \delta_2, \Delta}\ \parr\,\mathrm{R} \qquad \frac{\Gamma, \gamma_1, \gamma_2\ \langle f \vdash_\tau g\rangle\ \Delta}{\Gamma, \gamma_1 \otimes \gamma_2\ \langle f \vdash_\tau g\rangle\ \Delta}\ \otimes\mathrm{L}$$

$$\frac{\Gamma\ \langle f \vdash_\tau g\rangle\ \delta_1, \Delta}{\Gamma\ \langle f \vdash_\tau g\rangle\ \delta_1 \oplus \delta_2, \Delta}\ \oplus\mathrm{R1} \qquad \frac{\Gamma\ \langle f \vdash_\tau g\rangle\ \delta_2, \Delta}{\Gamma\ \langle f \vdash_\tau g\rangle\ \delta_1 \oplus \delta_2, \Delta}\ \oplus\mathrm{R2}$$

$$\frac{\Gamma, \gamma_1\ \langle f \vdash_\tau g\rangle\ \Delta}{\Gamma, \gamma_1 \& \gamma_2\ \langle f \vdash_\tau g\rangle\ \Delta}\ \&\mathrm{L1} \qquad \frac{\Gamma, \gamma_2\ \langle f \vdash_\tau g\rangle\ \Delta}{\Gamma, \gamma_1 \& \gamma_2\ \langle f \vdash_\tau g\rangle\ \Delta}\ \&\mathrm{L2}$$

$$\frac{\Gamma, \delta\ \langle f \vdash_\tau g\rangle\ \Delta}{\Gamma\ \langle f \vdash_\tau g\rangle\ \delta^\perp, \Delta}\ {}^\perp\mathrm{R} \qquad \frac{\Gamma\ \langle f \vdash_\tau g\rangle\ \gamma, \Delta}{\Gamma, \gamma^\perp\ \langle f \vdash_\tau g\rangle\ \Delta}\ {}^\perp\mathrm{L}$$

Environment Copying Rules:

$$\frac{\Gamma\ \langle f_l \vdash_{\tau\cdot l} g_l\rangle\ \delta_1, \Delta \quad \Gamma\ \langle f_r \vdash_{\tau\cdot r} g_r\rangle\ \delta_2, \Delta}{\Gamma\ \langle f_l \&_\tau f_r \vdash_\tau g_l \&_\tau g_r\rangle\ \delta_1 \& \delta_2, \Delta}\ \&\mathrm{R}$$

$$\frac{\Gamma, \gamma_1\ \langle f_l \vdash_{\tau\cdot l} g_l\rangle\ \Delta \quad \Gamma, \gamma_2\ \langle f_r \vdash_{\tau\cdot r} g_r\rangle\ \Delta}{\Gamma, \gamma_1 \oplus \gamma_2\ \langle f_l \&_\tau f_r \vdash_\tau g_l \&_\tau g_r\rangle\ \Delta}\ \oplus\mathrm{L}$$

Environment Splitting Rules:

$$\frac{\Gamma_1\ \langle f_l \vdash_{\tau\cdot l} g_l\rangle\ \delta_1, \Delta_1 \quad \Gamma_2\Gamma, \gamma_2\ \langle f_r \vdash_{\tau\cdot r} g_r\rangle\ \delta_2, \Delta_2}{\Gamma_1, \Gamma_2\ \langle f_l \otimes_\tau f_r \vdash_\tau g_l \parr_\tau g_r\rangle\ \delta_1 \otimes \delta_2, \Delta_1, \Delta_2}\ \otimes\mathrm{R}$$

$$\frac{\Gamma_1, \gamma_1\ \langle f_l \vdash_{\tau\cdot l} g_l\rangle\ \Delta_1 \quad \Gamma_2, \gamma_2\ \langle f_r \vdash_{\tau\cdot r} g_r\rangle\ \Delta_2}{\Gamma_1, \Gamma_2, \gamma_1 \parr \gamma_2\ \langle f_l \otimes_\tau f_r \vdash_\tau g_l \parr_\tau g_r\rangle\ \Delta_1, \Delta_2}\ \parr\,\mathrm{L}$$

Passive Formula Rules:

$$\frac{\Gamma\ \langle f \vdash_{\tau\cdot l} g\rangle\ \Delta}{\Gamma, A_i\ \langle f \vdash_\tau g \otimes_\tau (i \mapsto \tau\cdot r)\rangle\ \Delta}\ \mathrm{resL} \qquad \frac{}{!\Gamma\ \langle (i \mapsto \tau) \vdash_\tau \perp\rangle\ A_i, ?\Delta}\ \mathrm{resR}$$

$$\frac{}{!\Gamma, A\ \langle \mathbf{1} \vdash_\tau \perp\rangle\ A, ?\Delta}\ \mathrm{Ax} \qquad \frac{}{!\Gamma\ \langle f \vdash_\tau f\rangle\ ?\Delta}\ \mathrm{Eq}$$

In the rules $\otimes$R and $\parr$L, the right-hand argument place of $\langle X \vdash_\tau Y\rangle$ can be occupied either by a function or by $\perp$; $\parr$ of two such items is only defined if at least one of them is $\perp$.

Table 3: Realisability Conditions

Definition 2. The root of a proof tree is tagged with $[]$. Single premise rules are tagged thus:

$$\frac{\Gamma' \vdash_\tau \Delta'}{\Gamma \vdash_\tau \Delta}$$

Twin-premise rules are tagged thus:

$$\frac{\Gamma' \vdash_{\tau \cdot l} \Delta' \quad \Gamma'' \vdash_{\tau \cdot r} \Delta''}{\Gamma \vdash_\tau \Delta}$$

We can thus tag a proof-tree inductively, starting from the root. Notice that, if two sequents are separated by single-premise rules, they have the same tag; it is easy to check that this non-uniqueness does not affect our uses of tags (i.e. any functions which we define with domain the set of tags are genuinely single-valued).

Components of X and Y are labelled with *trajectories*, which are defined as follows.

Definition 3. A *trajectory* is a binary tree whose edges and root are labelled with tags.

We will draw trajectories thus, with the root at the bottom (here T_l and T_r are subtrees):

$$\begin{tikzcd} \mathbb{T} = & T_l & & T_r \\ & & \tau \arrow[ul, dash, "\tau_l"] \arrow[ur, dash, "\tau_r"'] & \end{tikzcd}$$

Here τ will be called the *root tag*, τ_l will be called the *left tag*, and τ_r the *right tag*. The tree T_l, labelled at its root with τ_l, will be called the *left subtrajectory*; the tree T_r, labelled at its root with τ_r, will be called the *right subtrajectory*. If we are only interested in the root tag, we can write our trajectories $\mathbb{T} = \begin{array}{c} T \\ | \\ \tau \end{array}$.

We define the following constructors:

Definition 4. The *initial trajectory* is the tree consisting of a single node, labelled with $[]$. Call the initial trajectory $\mathbb{I}$.

Definition 5. If τ is a tag and $\mathbb{T}$ is a trajectory, we define $\tau \cdot \mathbb{T}$ by $\tau \cdot \begin{array}{c} T \\ | \\ \tau' \end{array} = \begin{array}{c} T \\ | \\ \tau \cdot \tau' \end{array}$.

Definition 6. If $\mathbb{T}_1$ and $\mathbb{T}_2$ are two trajectories, then $\mathbb{T}_1 + \mathbb{T}_2$ is defined by

$$\begin{array}{c} T_1 \\ | \\ \tau_1 \end{array} + \begin{array}{c} T_2 \\ | \\ \tau_2 \end{array} = \begin{tikzcd} T_1 & & T_2 \\ & {[]} \arrow[ul, dash, "\tau_1"] \arrow[ur, dash, "\tau_2"'] & \end{tikzcd}.$$

Now we define the relation $\triangleleft$ as follows:

Definition 7. For trajectories $\mathbb{T}, \mathbb{T}_l$ and $\mathbb{T}_r$, and tags τ and τ', we have:

1. $\tau \triangleleft \mathbb{I}$ iff $\tau = []$;
2. $\tau \cdot \tau' \triangleleft \tau \cdot \mathbb{T}$ iff $\tau' \triangleleft \mathbb{T}$;

3. $l \cdot \tau \lhd \mathbb{T}_l + \mathbb{T}_r$ iff $\tau \lhd \mathbb{T}_l$;

4. $r \cdot \tau \lhd \mathbb{T}_l + \mathbb{T}_r$ iff $\tau \lhd \mathbb{T}_r$.

In pictures, $\lhd$ looks like this:

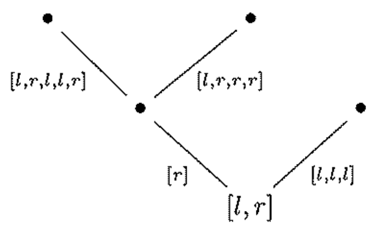
- $\tau \lhd \begin{array}{c} T \\ | \\ \tau \cdot \tau' \end{array}$;

- $\tau \cdot l \cdot \tau' \lhd \begin{array}{ccc} T_l & & T_r \\ {\scriptstyle \tau_l} \diagdown & & \diagup {\scriptstyle \tau_r} \\ & \tau & \end{array}$ iff $\tau' \lhd \begin{array}{c} T_l \\ | \\ \tau_l \end{array}$;

- $\tau \cdot r \cdot \tau' \lhd \begin{array}{ccc} T_l & & T_r \\ {\scriptstyle \tau_l} \diagdown & & \diagup {\scriptstyle \tau_r} \\ & \tau & \end{array}$ iff $\tau' \lhd \begin{array}{c} T_r \\ | \\ \tau_r \end{array}$.

Thus, τ lies on T iff we use it up in defining a path on T, starting at the root label, using initial segments to traverse edges with those labels, and, whenever we come to a node, using the head element (either l or r) to branch appropriately. Notice that the tags τ behave like queues: as we build the prooftree, we push elements onto the back of the tags involved, whereas, when we follow the trajectory of a proposition in a completed proof-tree, we pop elements off the front of a tag.

Example 1. Consider the following trajectory:



If we call this trajectory $\mathbb{T}$, then we have $[l] \lhd \mathbb{T}$, $[l,r,r,l,l] \lhd \mathbb{T}$, $[l,r,r,l,r] \not\lhd \mathbb{T}$, $[l.r,l,r,r,l] \lhd \mathbb{T}$, $[l,r,l,r,r,r] \not\lhd \mathbb{T}$, $[l,r,l,r,r,l,l,l,l] \not\lhd \mathbb{T}$.

We will use trajectories to mark the paths of components of the state vectors during a proof. More precisely:

Definition 8. Let $\overrightarrow{A} = \bigotimes_{i \in \mathcal{I}} A_i$ be a "state vector", let f be a partial function defined on I with values in the space of trajectories, and let τ be a arbitrary tag. Then let

- $(\mathcal{I}, f) /\!\!/ \tau = \{ i \in \mathcal{I} | \tau \lhd f(i) \}$, and let
- $(\overrightarrow{A}, f) /\!\!/ \tau = \bigotimes_{i \in (\mathcal{I}, f) /\!\!/ \tau} A_i$.

We will be considering state vectors together with such partial functions; for a node τ in a prooftree, $\overrightarrow{A} /\!\!/ \tau$ will be the subvector of A that takes part in the sequent τ. Now we want to show how we can synthesise functions like f from their restrictions $f /\!\!/ \tau$, where τ is a leaf of the prooftree.

First we define a couple of initial functions:

Definition 9. • $\mathbf{1}$ is the everywhere undefined partial function on $\mathcal{I}$. (We write it $\mathbf{1}$ because it represents the nullary tensor product, i.e. $\mathbf{1}$.)

- If $i \in I$ and if τ is a trajectory, then $i \mapsto \tau$ is the function which maps i to τ and is elsewhere undefined.

Then we define the following operations on functions:

Definition 10. Let f and g be partial functions defined on $\mathcal{I}$. Then

- $f \otimes_\tau g$ is defined iff
  - there is no i for which both $f(i)$ and $g(i)$ are defined, and
  - if $f(i)$ is defined, then $f(i) = \tau \cdot l \cdot f'(i)$, for some $f'(i)$, and
  - if $g(i)$ is defined, then $g(i) = \tau \cdot r \cdot g'(i)$ for some $g'(i)$.

  In this case, we let $f \otimes_\tau g(i) = f(i)$ or $g(i)$ whenever either one of them makes sense.

- $f \&_\tau g$ is defined iff
  - for all i, $f(i)$ is defined iff $g(i)$ is defined,
  - if $f(i)$ is defined, then $f(i) = \tau \cdot l \cdot f'(i)$, and
  - if $g(i)$ is defined, then $g(i) = \tau \cdot r \cdot g'(i)$.

  In this case, we let $f \&_\tau g(i) = \tau \cdot (f'(i) + g'(i))$ whenever this makes sense.

We now have

Proposition 4. *1.* $(\overrightarrow{A}, \mathbf{1}) /\!\!/ \tau = \mathbf{1}$;

2. $(\overrightarrow{A}, i \mapsto \tau) /\!\!/ \tau = A_i$;

3. If $f \otimes_\tau g$ *is defined, then* $(\overrightarrow{A}, f \otimes_\tau g) /\!\!/ \tau = ((\overrightarrow{A}, f) /\!\!/ \tau \cdot l) \otimes ((\overrightarrow{A}, g) /\!\!/ \tau \cdot r)$;

4. if $f \&_\tau g$ *is defined, then* $(\overrightarrow{A}, f \&_\tau g) /\!\!/ \tau = ((\overrightarrow{A}, f) /\!\!/ \tau \cdot l) = ((\overrightarrow{A}, g) /\!\!/ \tau \cdot r)$.

Proof The first two are obvious. For the third, we need to show that $(\mathcal{I}, f \otimes_\tau g) /\!\!/ \tau = ((\mathcal{I}, f) /\!\!/ \tau \cdot l) \coprod ((\mathcal{I}, g) /\!\!/ \tau \cdot r)$, where $\coprod$ is multiset union or disjoint union, depending on taste.[4] For this, we have to show that, for each i, $\tau \lhd f \otimes_\tau g(i)$ iff $\tau \cdot l \lhd f(i)$ or $\tau \cdot r \lhd g(i)$, but never both. However, whenever it is defined, $f(i) = \tau \cdot l \cdot f'(i)$, so whenever $f(i)$ is defined, $\tau \cdot l \lhd f(i)$; similarly for g and $f \otimes_\tau g$. But, by the preconditions for the definition of $\otimes_\tau$, $f \otimes_\tau g(i)$ is defined iff $f(i)$ or $g(i)$ are defined, and the latter two are never true simultaneously. The fourth is similar. $\square$

[4]This depends on whether we allow the indexing item $\mathcal{I}$ to be a set or a multiset; nothing hangs on it.

The purpose of this machinery is to be able to handle the passive components of sequents by synthesising them starting from the leaves of a proof tree. We have seen (p. 16) that the proof tree can be restructured so that its leaves are instances of the passive formula rules; the corresponding rules in Table 3 show how to assign functions to these rules. We can then use the other rules in the table to assign functions to the rest of the nodes of the prooftree. From these functions, we can use the $/\!\!/$ operation to recover the appropriate state vectors, X and Y, for each sequent in the tree.

This is still a little cumbersome, however; we have to work with a multiplicity of functions. We can avoid this by using the following stability properties:

Proposition 5. *Let f and g be the functions attached to a node τ, and let f' and g' be the fuctions attached to τ', where τ' is an ancestor of τ (i.e. $\tau = \tau' \cdot \tau''$, for some τ''. Then $(\overrightarrow{X}, f)/\!\!/\tau = (\overrightarrow{X}, f')/\!\!/\tau$, and similarly for Y, g and g'.*

Proof We prove this by induction on the length of τ'', so we can assume that $\tau = \tau' \cdot l$ or $\tau = \tau' \cdot r$. Then the uppermost node labelled with τ must be the conclusion of a two-premise rule; if it is an environment copying rule, then f' must be obtained from f and another function by means of $\&_{\tau'}$, whereas if it is an environment splitting rule (or resL) then f' must be obtained from f and another function by means of $\otimes_{\tau'}$. These are the only two-premise rules, so it suffices to verify the proposition in each of these cases.

Consider, for example, the case when $\tau = \tau' \cdot l$ and where $f' = f \&_{\tau'} h$. We have to show that, for all i, $\tau \lhd f(i)$ iff $\tau \lhd f'(i)$. Since $f \&_{\tau'} h$ is defined, we must have $f'(i)$ is defined iff $f(i)$ is defined iff $h(i)$ is defined. These are the only values of i we need consider. Fix one such, and let $f(i) = \tau \cdot l \cdot \mathbb{T}_1$, $h(i) = \tau \cdot r \cdot \mathbb{T}_2$; they must have this form since $f \&_{\tau'} h$ is defined. So clearly $\tau \lhd f(i)$ and $\tau \lhd f'(i)$. The other cases are similar. $\square$

We can, then, work with only one f and one g, namely those attached to the root of the proof tree. Given a resolution proof, we have a global object, consisting of the two state vectors X and Y, and the two functions f and g; these can be considered as a realiser of the original Lygon query (i.e. the active formulae of the original sequent). That is, for each connective, we have clauses of the form

$$\mathcal{R} \text{ realises } A \otimes B \text{ iff } \mathcal{R}' \text{ realises } A \text{ and } \mathcal{R}'' \text{ realises } B,$$

for $\mathcal{R}$ recursively defined in terms of $\mathcal{R}'$ and $\mathcal{R}''$, and we also have suitable clauses giving the realisers for atoms.

6.2 Proof Search

We have a proof search algorithm, as follows:

Definition 11. A realisability proof search will proceed as follows:

1. apply active rules until no more compound active formulae are left;
2. apply resL to the active atoms on the left-hand side of the sequent until no more atoms are left;
3. assign functions to the leaves;

4. assign functions inductively to the rest of the tree until we assign a pair of functions f and g to the root.

Notice that this procedure may fail at any stage after the first: we may not be able to apply resL to a left active atom because there may not be a corresponding atom in Y, the leaves may not be axiom sequents so we may not be able to assign functions to them, and the functions we assign may not satisfy the preconditions for $\otimes_\tau$ and $\&_\tau$ to be defined.

However, we can prove that this procedure is sound and complete.

Theorem 1 (Soundness). *If the above procedure assigns a tree, together with a pair of functions f and g, to our original sequent, then the sequent is valid.*

Proof Assign, to a node of the tree labelled with τ, the vectors $(X, f) /\!\!/ \tau$ and $(Y, g) /\!\!/ \tau$. We will show that the resulting tree is a resolution proof of the sequent. We have to show that each node is the correct application of one of the sequent calculus rules, and that all of the leaves correspond to axioms.

The leaves can only be assigned functions by the rules resR, Ax, and Eq, and, after the leaves are assigned state vectors, these all become instances of axiom rules.

As for the other nodes, it is clear that the single premises are correct applications of the rules. What is necessary for correctness is that these rules should have the same state vectors assigned to premises and conclusion; however, the vectors only depend on the tags, and for single premise rules, premise and conclusion have the same tags.

For the environment copying rules, suppose that the conclusion of such a rule has tag τ; the premises will then be tagged with $\tau \cdot l$ and $\tau \cdot r$. Let the corresponding vectors be X_τ, $X_{\tau \cdot l}$, and so on; we have to show that $X_\tau = X_{\tau \cdot l} = X_{\tau \cdot r}$, and similarly for the Y's. Our functions f and g were built up inductively over the tree; let f'_l be the function constructed at the node $\tau \cdot l$, f'_r the function at the node $\tau \cdot r$, and f' be the function at the node τ. By definition, we have $f' = f'_l \&_\tau f'_r$, and so, by Proposition 4, we have $(X, f') /\!\!/ \tau = (X, f'_l) /\!\!/ \tau \cdot l = (X, f' \cdot r) /\!\!/ \tau \cdot r$. However, by Proposition 5, $(X, f') /\!\!/ \tau = (X, f) /\!\!/ \tau$, $(X, f'_l) /\!\!/ \tau \cdot l = (X, f) /\!\!/ \tau \cdot l$, $(X, f'_r) /\!\!/ \tau \cdot r = (X, f) /\!\!/ \tau \cdot r$, which establishes what we want; the proof for the Y's is similar.

The proof for the environment splitting rules is exactly the same. □

Theorem 2 (Completeness). *The above proof search algorithm is complete: if there is a proof of the sequent $\Gamma_0, \Gamma \langle X \vdash Y \rangle \Delta$, then the algorithm will find it.*

Proof Suppose that there is a proof of the sequent; we will permute inferences in the proof so that it attains the desired form. We first use Proposition 2 to permute applications of the passive formula rules above the active formula rules, apart from instances of $\otimes$ L occurring just before an environment-splitting rule. The first stage of the algorithm will then find this part of the proof tree. The sequents remaining when all of the active formula rules have been applied will have (at most) active linear atoms on both sides; since the sequent is provable, these sequents must have one of the forms given in Lemma 1. These sequents can all be proved by applying resL first and then using the other resolution rules. Our algorithm will correctly find a proof tree of the desired shape; we now show (by induction over the tree) that the assignment of functions to the nodes is successful, and that, when we assign corresponding state vectors to the nodes, we get a correct proof tree. □

We have, then, a sound and complete proof search procedure which can deal with situation calculus proofs in a computationally efficient way.

A Examples

This section contains several standard AI examples, handled using the formalism that we have developed. Since this level of the formalism deals only with simple changes and persistence, we will only be dealing with examples that can be solved in this way (in particular, there will be no ramification). The examples are mostly from John Bell's compilation [3].

A.1 The Yale Shooting Problem

The scenario is:

> At time 1 a gun is loaded and pointed at Fred. Nothing relevant happens at time 2. At time 3 the gun is fired. [3, Example 3.4]

This can be solved by the following Lygon code. The program (i.e. $!\Gamma_0$) consists of the following background laws, which define the effects of the two actions (loading and shooting):

$$\begin{aligned} !(\mathsf{load} &\leftarrow \mathsf{unloaded}^{\perp} \parr \mathsf{loaded}) & (4)\\ !(\mathsf{shoot} &\leftarrow (\mathsf{alive} \otimes \mathsf{loaded})^{\perp} \parr (\mathsf{dead} \otimes \mathsf{unloaded})) & (5) \end{aligned}$$

With this $!\Gamma_0$ (together with the code for the metainterpreter) we solve the goal (cf. [14]):

$$\langle \mathsf{alive} \otimes \mathsf{unloaded} \mid \mathsf{load} \vdash X\rangle \;\&\; \langle X \mid \mathbf{1} \vdash Y\rangle \;\&\; \langle Y \mid \mathsf{shoot} \vdash Z\rangle$$

and we find the following solution:

$$\begin{aligned} X &= \mathsf{alive} \otimes \mathsf{loaded}\\ Y &= \mathsf{alive} \otimes \mathsf{loaded}\\ Z &= \mathsf{dead} \otimes \mathsf{unloaded}. \end{aligned}$$

There are no other solutions.

A.2 The Stanford Murder Mystery

The scenario is:

> In this variation of the Y[ale] S[hooting] P[roblem], Fred is alive at time 1. A shoot action occurs at time 1. Nothing is known to happen at time 2. Fred is known to be dead at time 3. When did Fred die, and was the gun loaded initially? [3, Example 4.5]

We use the same code as for the Yale Shooting Problem, and solve the goal

$$\langle \mathsf{alive} \otimes X \mid \mathsf{shoot} \vdash Y \rangle \mathbin{\&} \langle Y \mid \mathbf{1} \vdash \mathsf{dead} \otimes Z \rangle$$

and we find the solution

$$\begin{aligned} X &= \mathsf{loaded} \\ Y &= \mathsf{dead} \otimes \mathsf{unloaded} \\ Z &= \mathsf{unloaded}. \end{aligned}$$

There are no other solutions.

Notice the extent of code re-use between the Yale Shooting Problem; not only is this economical, but it is intuitively satisfying (it makes clear that prediction and explanation are applications of the same theory with a different temporal direction; by contrast, Bell [3, Section 4] has to use different theories for prediction and explanation, and has no way of coming to terms with the intuition that prediction and explanation must use the same knowledge of the world.)

A.3 The Russian Shooting Problem

The scenario is

> Fred plays Russian Roulette. He loads the revolver with a single bullet, spins the magazine, puts the gun to his head and shoots. Is he dead as a result?

It is important to be sensitive to language here: 'shoots' cannot be right, because it is what Ryle calls a "success word", that is, it is an action word whose use presupposes the success of the action referred to.

> There is another class of episodic words which, for our purposes, merit special attention, namely the class of episodic words which I have elsewhere labelled 'achievement words', 'success words' or 'got it words', together with their antitheses the 'failure words' or 'missed it words'. These are genuine episodic words, for it is certainly proper to say of someone that he scored a goal at a particular moment, repeatedly solved anagrams, or was quick to see the joke or find the thimble. Some words of this class signify more or less sudden climaxes or dénouments; others signify more or less protracted proceedings. ...
>
> The verbs with which we ordinarily express these gettings and keepings are active verbs, such as 'win', 'unearth', 'find', 'cure', 'convince', 'prove', 'cheat', 'unlock', 'safeguard' and 'conceal'; and this grammatical fact has tended to make people, with the exception of Aristotle, oblivious to the differences of logical behaviour between verbs of this class and other verbs of activity or process. The differences, for example, between kicking and scoring, treating and healing, hunting and finding, clutching and holding fast, listening and hearing, looking and seeing, travelling and arriving, have been construed, if they have been noticed at all, as differences between

> co-ordinate species of activity or process, when in fact the differences are of another kind. ...
>
> One big difference between the logical force of a task verb and that of a corresponding achievement verb is that in applying an achievement verb we are asserting that some state of affairs obtains over and above that which consists in the performance, if any, of the subservient task activity. For a runner to win, not only must he run but also his rivals must be at the tape later than he; for a doctor to effect a cure, his patient must both be treated and be well again; for the searcher to find the thimble, there must be a thimble in the place he indicates at the moment he indicates it; and for the mathematician to prove a theorem, the theorem must be true and follow from the premises from which he tries to show that it follows. An autobiographical account of the agent's exertions and feelings does not by itself tell whether he has brought off what he was trying to bring off. ...
>
> That is why we can significantly say that someone has aimed in vain or successfully, but not that he has hit the target in vain or successfully; that he has treated the patient assiduously or unassiduously, but not that he has cured him assiduously or unassiduously. [19, pp. 149ff.]

Thus, shoot, which was analysed above (5) as

$$!(\mathsf{shoot} \quad \leftarrow \quad (\mathsf{alive} \otimes \mathsf{loaded})^{\perp} \parr (\mathsf{dead} \otimes \mathsf{unloaded}))$$

cannot be part of a deduction unless we have loaded at the appropriate stage for it to consume. (It also presupposes that the victim is alive beforehand; strictly speaking, this is an analysis of the success word 'shoot dead'. We cannot shoot dead someone who is already dead.[5]) This was appropriate for the stories we used it in.

However, in this case the appropriate concept is not *shooting*, but *trying to shoot* (or, arguably, something like *aiming and pulling the trigger*): an action which results in a shooting when the preconditions are satisfied, but not otherwise. It is, in Ryle's terminology, the task verb which underlies the achievement verb 'to shoot'. The analysis of the task verb is

$$\begin{aligned} !(\mathsf{try\_shoot} \quad \leftarrow \quad & ((\mathsf{alive} \otimes \mathsf{loaded})^{\perp} \parr (\mathsf{dead} \otimes \mathsf{unloaded})) \\ & \&(\mathsf{unloaded}^{\perp} \parr \mathsf{unloaded})) \end{aligned}$$

We add to this a clause for spinning the magazine

$$!(\mathsf{spin} \quad \leftarrow \quad \mathbf{1} \& (\mathsf{loaded}^{\perp} \parr \mathsf{unloaded}))$$

(so that spinning the magazine will nondeterministically unload the gun or leave it loaded). We then solve the goal

$$\langle \mathsf{alive} \otimes \mathsf{unloaded} \mid \mathsf{load} \vdash X \rangle \;\&\; \langle X \mid \mathsf{spin} \vdash Y \rangle \;\&\; \langle Y \mid \mathsf{try\_shoot} \vdash Z \rangle$$

[5] Cf. [28, p. 43]: "Only when the door *is* closed can it become opened. One cannot open an open door."

and we get nondeterministically the two solutions:

$$\begin{aligned} X &= \mathsf{alive} \otimes \mathsf{loaded} \\ Y &= \mathsf{alive} \otimes \mathsf{loaded} \\ Z &= \mathsf{dead} \otimes \mathsf{unloaded} \end{aligned}$$

and

$$\begin{aligned} X &= \mathsf{alive} \otimes \mathsf{loaded} \\ Y &= \mathsf{alive} \otimes \mathsf{unloaded} \\ Z &= \mathsf{alive} \otimes \mathsf{unloaded}. \end{aligned}$$

What should be noticed here is the close correspondence between the philosophical analysis and the logic: the exacting nature of linear logic means that we must analyse the scenario more sensitively, in order to find a formalism that works.

A.4 The Stolen Car Problem

The scenario is

> You park your car in the lot at time 1. You discover that it is gone at time 3. You conclude that it could have been stolen at any time in between.

We analyse steal as

$$!(\mathsf{steal} \quad \leftarrow \quad \mathsf{have}^{\perp} \parr \mathsf{not\_have})$$

and then we solve the goal

$$\langle \mathsf{have} \mid \mathsf{steal}\&\mathbf{1} \vdash X \rangle \,\&\, \langle X \mid \mathsf{steal}\&\mathbf{1} \vdash \mathsf{not\_have} \rangle \ .$$

This has two solutions:

$$X \quad = \quad \mathsf{not\_have}$$

and

$$X \quad = \quad \mathsf{have}$$

corresponding to thefts at time 1 or time 2, respectively.

Note that the clauses steal&**1** will each nondeterministically be replaced by either steal or **1**, using the rules & L1 and & L2. Thus, *prima facie* we have four possibilities; however, one of them (two successive thefts) is ruled out because we have analysed steal as a success word, and one cannot steal a car which has already been stolen, whereas another (two successive non-thefts) is ruled out because it does not produce the correct final state.

B How to Fail

Let us return to Desideratum 1. There are two ways of not fulfilling this desideratum, and they can be demonstrated on a variety of different implementations. Although we have been working proof-theoretically, the desideratum makes sense in a wider context; one needs some sort of concept of a transformation between situations, and one also wants to know what the identity transformations are. Furthermore, one needs some sort of "product" (corresponding to the linear $\otimes$) for composing situations. The desideratum will then state that, if the state vector, or part of the state vector, is unchanged by an event, then that subvector should be represented by the identity transformation; a general transformation should be the product of the identity on the unchanged part of the state and some other transformation on the changed part.

In any event, once we have such an analysis, and if it satisfies the desideratum, then we can use this to design an efficient implementation: identities in the model can be implemented by identities, that is, we can do a great deal of structure sharing between the original state vector and its changed version. This is especially relevant for languages like Prolog, since Prolog interpreters (or a majority of them) perform a great deal of structure sharing; [16, pp. 73ff.] indeed, the introduction of this approach was one of the things which made Prolog into a workable programming language.

This applies to proof-theoretic formulations, using the Curry-Howard isomorphism (formulae are objects of a suitable category, proofs its morphisms); but the same sort of analysis can also be used with model-based algorithms such as we shall be discussing in this section. These algorithms are given a theory T consisting of Horn clauses, and, starting with a model at t, they compute a model at $t+1$ which

- extends the model at t,
- satisfies the theory in the sense that, if the antecedent of a sentence of the theory is satisfied by the model at t, the consequent must be satisfied by the theory at $t+1$, and
- is minimal subject to these conditions.

We can consider the models to be objects of our category, and theories to be the morphisms. The "product" will be union of models; if we define our morphisms suitably, the product will actually be direct sum in our category, but not much hangs on this.

B.1 Types of Failure

The desideratum, then, can fail in several ways. One is to find a proof – or, more generally, a morphism – which is genuinely of the above form, that is, which treats separately the transformed and the untransformed parts of the state vector, but not to use the identity proof for $Z \to Z$. We may further classify the failures by the complexity of applying the inference to Z.

There is one notable source for such failures: it comes from approaches (such as the model-building approach) where we consider state changes as deductions, but use classical logic (or slight modifications thereof, such as Kleene's three-valued logic). Suppose we have a proposition P which is true on Monday, but false on Tuesday. If our theory has a proposition (call it A) which yields $\neg P$, given P, then – because of the monotonicity of classical logic – A and P entail both P and $\neg P$. In order to avoid this contradiction, one generally works, not with bare propositions such as P, but with propositions indexed either by situations or by times. But now we have lost any chance of having a deduction of the form we want: even if a proposition Q has the same truth value on Monday and Tuesday, $Q(\mathsf{Tuesday})$ and $Q(\mathsf{Monday})$ are now *different* propositions, and the deduction of one from the other will contribute to the complexity of the deduction. We do not share anything between the state at t and $t+1$: a new copy must be made. The whole world must be created anew with each new instant. Although this approach may be amusingly reminiscent of the philosophies of ibn Rushd and of Malebranche, as a programming strategy it is daft beyond belief. (Cf. [11], [13, p. 2])

B.2 Linear Failures

The theory given in Table 4 exhibits the usual sort of failure, and, for typical problems (that is, those with the state vector large and with a small number of simultaneous movements) the time complexity is linear in the size of the state vector, regardless of the size of the state changes. Notice that, although it uses non-identity transformations for the unchanged part, they are essentially given by the inferences

$$\mathsf{holds}(s)(t) \wedge \bullet\,\mathsf{affected}(s)(t) \quad \rightarrow \quad \mathsf{holds}(s)(t+1); \tag{6}$$

that is, they only involve propositions which are directly related to the state. If there is no change, then the proposition $\mathsf{affected}(s)(t)$ has truth-value $\perp$, and thus doesn't occur in the model. The transformation between two states, then, can be decomposed into the transformation between the changed parts and the application of (6) to the unchanged part. The latter application accounts for the linear component of the complexity.

B.3 Quadratic Failures

It is possible, however, to do substantially worse than merely a linear failure of Desideratum 1. The implementation in Table 5 does that; the maps between the unchanged parts of the state are handled by the inferences

$$\mathsf{fluent}(s) \wedge s(t) \wedge \circ s(t+1) \quad \rightarrow \quad s(t+1).$$

Now in order for these to perform correctly (and in particular to avoid incorrectly applying identity inferences to the positions of moved blocks) we have to add, to the state description, propositions of the form $\neg\,\mathsf{at}(b,l)$, which say that a given block is not at a given position; these can then be used to block the application of the persistence rule. However, the presence of these extra propositions means that the size of the state description is now quadratic in the size of the situation (more precisely, it is linear in the number of objects and linear in the number of positions).

Notice that, by contrast with the merely linear failure, this failure exhibits some sort of failure of decomposability. The state description for the implementation of Table 4 consists only of propositions of the form $\mathsf{at}(b,l)$, one for

$$\begin{array}{rcl}
\mathsf{involves}(\,\mathsf{moved}(b,l1,l2),\ \mathsf{at}(b,l1)) & & \\
\mathsf{holds}(f)(t) \wedge \mathsf{occurs}(e)(t) \wedge \mathsf{involves}(e,f) & \rightarrow & \mathsf{affected}(f)(t) \\
\mathsf{postconds}(\,\mathsf{moved}(b,l1,l2)(t) & \rightarrow & \mathsf{holds}(\,\mathsf{at}(b,l2)(t) \\
\mathsf{holds}(s)(t) \wedge \bullet\,\mathsf{affected}(s)(t) & \rightarrow & \mathsf{holds}(s)(t+1) \\
\mathsf{holds}(\,\mathsf{at}(b,l1))(t) & \rightarrow & \mathsf{preconds}(\,\mathsf{moved}(b,l1,l2))(t) \\
\mathsf{preconds}(e)(t) \wedge \mathsf{occurs}(e)(t) & & \\
\wedge\bullet\,\mathsf{qualified}(e)(t) & \rightarrow & \mathsf{postconds}(e)(t+1)
\end{array}$$

This theory updates the positions of blocks (given by propositions such as $\mathsf{holds}(\,\mathsf{at}(b,l1))(t))$ after movements (given by propositions such as $\mathsf{occurs}(\,\mathsf{moved}(b,l1,l2))(t))$. Propositions are temporally indexed, and the deductive system is that given by Kleene's three-valued logic together with the additional connectives $\bullet$ and $\circ$ whose truth tables are given by:

P	$\bullet P$	$\circ P$
T	F	T
$\perp$	T	T
F	T	T

The procedure is model-based rather than proof-theoretic: that is, a state at time t is given by those elements of a model of the theory which have temporal index t. Given a state at t, the corresponding state at $t+1$ can be found by forward chaining using the theory.

Table 4: A Linear Failure

$$\begin{array}{rcl}
 & & \mathsf{fluent}(\,\mathsf{at}(x,y))) \\
\mathsf{postconds}(\,\mathsf{moved}(b,l1,l2))(t) & \rightarrow & \mathsf{at}(b,l2)(t) \\
\mathsf{at}(b,l1)(t) \wedge \mathsf{unequal}(l1,l2) & \rightarrow & \neg\,\mathsf{at}(b,l2)(t) \\
\mathsf{fluent}(s) \wedge s(t) \wedge \circ s(t+1) & \rightarrow & s(t+1) \\
\mathsf{at}(b,l1)(t) & \rightarrow & \mathsf{preconds}(\,\mathsf{moved}(b,l1,l2))(t) \\
\mathsf{preconds}(e)(t) \wedge \mathsf{occurs}(e)(t) & & \\
\wedge\bullet\,\mathsf{qualified}(e)(t) & \rightarrow & \mathsf{postconds}(e)(t+1)
\end{array}$$

This theory solves the block movement problem, using the same deductive system as that in Table 4.

Table 5: A Quadratic Failure

each block. The total set of positions is not represented explicitly (provided we have a position for each block). Correspondingly, there is only one meaningful decomposition of the situation, that is, by taking apart the set of blocks; tensor product is, quite simply, union of the corresponding state descriptions. And our state transformations, though they do not actually have the form (1), can nevertheless be represented as a tensor product of mappings applied to atomic situations, i.e. situations which consist of a single proposition $\mathsf{at}(b, l)$. Although these mappings are not the identity, they are nevertheless quite straightforward, and the cost is linear.

However, the situation is otherwise with the implementation in Table 5. Although we can take state descriptions apart by decomposing the set of objects, we are still left with irreducible state descriptions of the form

$$\begin{array}{r} \mathsf{at}(b, l_1) \\ \neg\,\mathsf{at}(b, l_2) \\ \neg\,\mathsf{at}(b, l_3) \\ \vdots \end{array} \tag{7}$$

which, although they contain appreciable structure (and can be quite large), nevertheless cannot be decomposed. We are, of course, tacitly putting some restriction here on what might count as a component situation and what might not (we don't want just any subset of the state description). However, if we want to apply this analysis to the situation, whatever we count as a component situation must be sufficient, of itself, to correctly decide on questions of persistence. That is, if we have a component, the morphism, when applied to the component, must produce the same result as when applied to the full model. Thus, for the theory of Table 5, a situation must at least contain, for each block that it deals with, a full list of the form (7).

B.4 Worse Complexity

It is worth observing that, in at least one situation, maximal indecomposability goes with extremely bad complexity. When, that is, we have a non-monotonic logic described simply by model preference and where we are compelled to search among models, specified globally, then that search has complexity $\Pi_2^p =$ co-NP$^{\mathrm{NP}}$ [6, Section 6.1], which is as bad as non-monotonic logic gets. This connection seems quite significant, since such a search among irreducible models can be directly represented by a quantified Boolean formula, and these formula are a standard way of finding problems which are complete for complexity classes such as Π_2^p.

B.5 Timings

We can, in any case, give some comparative results which show that these things make a difference. Table 6 shows the differences between the implementations we have been discussing, when applied to instances of the block movement problem. They are all specified by three parameters: the number of blocks, the number of locations, and the number of blocks which are moved. The various settings (which correspond to, as it were, domestic situations, and which therefore ought to be AI problems of realistic size) are:

	Lygon	Table 4	Table 5
Chess	8.35	650	16876.6
Backgammon (easy)	65.6	991.7	10867.0
Backgammon (hard)	614	1683.4	19513.4
Kitchen	114.56	28418.3	2200000[a]

All times in milliseconds.

[a]Estimated value.

Table 6: Comparative Timings

Chess The problem of moving one piece on a chessboard; that is, there are 32 blocks, 64 locations, and one block is moved.

Backgammon (easy) A backgammon board has 24 files, and there are 30 pieces. A varying number of pieces is moved; on average, about 5 are moved at once. It's rather arbitrary how many locations we assign, but if we assign two locations per file, this seems realistic. So we get 30 pieces, 48 locations, and 5 pieces moved.

Backgammon (hard) One could theoretically move all 15 of one's pieces at once, though this is unlikely. In this case, we would have 30 pieces, 48 locations, and 15 pieces moved.

A Kitchen My kitchen has (at a rough count) 500 objects in it, of which about 10 would be used in a typical cooking operation. So we can consider this problem to have 500 pieces, 550 locations, and 10 pieces moved.

The results, then, bear out the analysis. We should notice that these problem situations are by no means excessively large or complex: they have the same sort of size as many everyday situations. Nevertheless, approaches which have been seriously recommended in AI perform extremely badly on problems of this size; by contrast, the Lygon metainterpreter performs well, and its performance scales well to large situations.

References

[1] Harold Abelson and Gerald Jay Sussman with Julie Sussman, *Structure and Interpretation of Computer Programs*, Cambridge MA: MIT Press 1985.

[2] Jon Barwise, email message to the linear mailing-list, 4 February 1992; available at `http://www.csl.sri.com/linear/mailing-list-traffic/www/07/mail_2.html`

[3] John Bell, "Prediction Theories and Explanation Theories", preprint.

[4] Wolfgang Bibel and Michael Thielscher, "Deductive Plan Generation", in Setsuo Arikawa and Klaus P. Jantke (eds.), *Algorithmic Learning Theory*

(The 4th International Workshop on Analogical and Inductive Inference AII 94, The 5th International Workshop on Algorithmic Learning Theory, ALT 94) *Lecture Notes in Artificial Intelligence* 872, Berlin: Springer 1994, pp. 2–5.

[5] Gerhard Brewka, *Non-Monotonic Reasoning*, Cambridge: Cambridge University Press 1991.

[6] Marco Cadoli and Marco Schaerf, "A Survey on Complexity Results for Non-Monotonic Logics", *Journal of Logic Programming* 17 (1993), pp. 127–166.

[7] Donald Davidson, "The Logical Form of Action Sentences", in Davidson, *Essays on Actions and Events*, Oxford: Clarendon 1980, pp. 105–122.

[8] Kerstin Eder, Steffen Hölldobler and Michael Thielscher, "An Abstract Machine for Reasoning about Situations, Actions, and Causality", in Roy Dyckhoff, Heinrich Herre, and Peter Schroeder-Heister (eds.), *Extensions of Logic Programming* (5th International Workshop, ELP 96), *Lecture Notes in Artificial Intelligence* 1050, Berlin: Springer 1996, pp. 137–151.

[9] Christoph Fouqueré and Jacqueline Vauzeilles, "Linear Logic and Exceptions", *Journal of Logic and Computation* 4 (1994), 859–875.

[10] Bertram Fronhöfer, "Linear Proofs and Linear Logic", in D. Pearce and G. Wagner (eds), *Logics in AI* (European Workshop JELIA 92), *Lecture Notes in Artificial Intelligence* 633, Berlin: Springer 1992, pp. 106–125.

[11] Bertram Fronhöfer, "Situation Calculus, Linear Connection Proofs and STRIPS-like Planning: An Experimental Comparison", in P. Miglioli, U. Moscato, D. Mundici and M. Ornaghi (eds.), *Theorem Proving with Analytic Tableaux and Related Methods* (5th International Workshop, TABLEAUX 96), *Lecture Notes in Computer Science* 1071, Berlin: Springer 1996, pp. 193–209.

[12] Gerd Grosse, Steffen Hölldobler and Josef Schneeberger, "Linear Deductive Planning", *Journal of Logic and Computation* 6 (1996), pp. 232–262.

[13] Steffen Hölldobler and Michael Thielscher, "Properties *versus* Resources: Solving Simple Frame Problems", Technische Hochschule Darmstadt Forschungsbericht AIDA–96–03 (1996).

[14] The Lygon home page, `http://www.cs.mu.oz.au/~winikoff/lygon/lygon.html`

[15] M. Masseron, C. Tollu and J. Vauzeilles, "Generating Plans in Linear Logic", in K.V. Nori and C.E. Veni Madhavan (eds.), *Foundations of Software Technology and Theoretical Computer Science* (Tenth Conference, Bangalore, India), *Lecture Notes in Computer Science* 472, Berlin: Springer 1990, pp. 63–75.

[16] Richard A. O'Keefe, *The Craft of Prolog*, Cambridge, MA: MIT 1990.

[17] David J. Pym and James A. Harland, "A Uniform Proof-Theoretic Investigation of Linear Logic Programming", *Journal of Logic and Computation* 4 (1994), 175–207.

[18] Ray Reiter, "Natural Actions, Concurrency and Continuous Time in the Situation Calculus", *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR'96)*, Cambridge MA., November 5-8, 1996.

[19] Gilbert Ryle, *The Concept of Mind*, London: Hutchinson 1949.

[20] Erik Sandewall, "The Range of Applicability of some Non-Monotonic Logics for Strict Inertia", *Journal of Logic and Computation* 4 (1994), 581–615.

[21] Erik Sandewall, *Features and Fluents*, Oxford: Oxford University Press 1994.

[22] Murray Shanahan, *Solving the Frame Problem* (unpublished MS).

[23] Richard Sorabji, *Necessity, Cause, and Blame: Perspectives on Aristotle's Theory*, London: Duckworth 1980.

[24] Michael Thielscher, "Computing Ramifications by Postprocessing", IJCAI 1995 pp. 1994–2000.

[25] A.S. Troelstra, "Tutorial on Linear Logic" in Peter Schroeder-Heister and Kosta Došen (eds), *Substructural Logics*, Oxford: Clarendon 1993, pp. 327–355.

[26] Michael Winikoff and James A. Harland, "Implementing the Linear Logic Programming Language Lygon", preprint.

[27] Michael Winikoff and James A. Harland, *Deterministic Resource Management for the Linear Logic Programming Language Lygon*, Technical Report 94/23, Department of Computer Science, University of Melbourne (1994).

[28] Georg Henrik von Wright, *Norm and Action: A Logical Enquiry*, London: Routledge and Kegan Paul, 1963.