ISSN 1369-1961

Department of Computer Science

Technical Report No. 741

GOLOG and Linear Logic Programming

Dr. G. White

QUEEN MARY
AND WESTFIELD COLLEGE
UNIVERSITY OF LONDON

January 1998

GOLOG and Linear Logic Programming*

Dr. G. White
Department of Computer Science
Queen Mary and Westfield College
University of London
London E1 4NS
email graham@dcs.qmw.ac.uk
url http://www.dcs.qmw.ac.uk/~graham

January 2, 1998

Abstract

We define a translation between the language GOLOG and a fragment of linear logic augmented with definitions; we prove bisimulation for this translation and finally suggest some extensions to GOLOG motivated by the translation.

Contents

*During the composition of this report, the author was paid by Project Dynamo, supported by the United Kingdom Engineering and Physical Sciences Research Council under grant number GR/K 19266. The views expressed in this paper are the author's own, and the principal investigators of the project – John Bell and Wilf Hodges – bear no responsibility for them.

0 Preliminaries

Levesque *et al.* [6] have defined a programming language, GOLOG, in order to reason about complex actions within the framework of the situation calculus. We build on previous work [17] and show how to translate this language into linear logic, suitably augmented.

0.1 Notation

Multisets will be written $\{\!|a, a, a, b, c, d, a|\!\}$, etc; $\emptyset$ will also be the empty multiset. Multiset union and intersection are $\sqcup$ and $\sqcap$.

0.2 Girard's Fixpoint Theorem

We will need to define some extra machinery, beyond the standard linear logic connectives; although we could do this in an *ad hoc* fashion, we will use Girard's Fixpoint Theorem, which provides a useful generic cut elimination theorem for the sort of extensions of linear logic that we will be considering.

We recall Girard's Fixpoint Theorem: see [4, 8, 13]. We quote the theorem in something like Girard's form:

Definition 1.

- A *right clause* is written $\dfrac{\vdash B}{\vdash h}\,\mathsf{df}$, where h, the *head*, is a linear logic term, and B, the *body*, is a linear logic formula without exponentials;
- A *left clause* is written $\dfrac{B \vdash}{h \vdash}\,\mathsf{df}$, where h, the *head*, is a linear logic term, and B, the *body*, is a linear logic formula without exponentials;

- A *definition* is a finite set of clauses.

Definitions give rise to sequent calculus rules. Suppose we have a definition, $\mathcal{D}$, and suppose (for simplicity) that no head of a left clause unifies with any head of a right clause. Then we define the following.

Definition 2. The rules corresponding to $\mathcal{D}$ are:

- $\dfrac{\Gamma \vdash F, \Delta}{\Gamma \vdash A, \Delta}$ where there is some right clause $\dfrac{\vdash B}{\vdash h}\,\mathsf{df}$ in $\mathcal{D}$ and a substitution ς such that $A = h\varsigma$, $F = B\varsigma$;
- $\dfrac{\Gamma, F_1 \vdash \Delta \quad \ldots \Gamma, F_k \vdash \Delta}{\Gamma, A \vdash \Delta}$ where $\left\{\dfrac{\vdash B_i}{\vdash h_i}\,\mathsf{df}\right\}_{i=1}^{k}$ are the right clauses such that h_i unifies with A, and where – if, for each i, ς_i is the most general unifier of h_i with A – $F_i = B_i\varsigma_i$;
- $\dfrac{\Gamma, F \vdash \Delta}{\Gamma, A \vdash \Delta}$ where there is some left clause $\dfrac{B \vdash}{h \vdash}\,\mathsf{df}$ in $\mathcal{D}$ and a substitution ς such that $A = h\varsigma$, $F = B\varsigma$;
- $\dfrac{\Gamma \vdash F_1, \Delta \quad \ldots \Gamma \vdash F_k, \Delta}{\Gamma \vdash A, \Delta}$ where $\left\{\dfrac{B_i \vdash}{h_i \vdash}\,\mathsf{df}\right\}_{i=1}^{k}$ are the left clauses such that h_i unifies with A, and where – if, for each i, ς_i is the most general unifier of h_i with A – $F_i = B_i\varsigma_i$.

Remark 1. We can think of the right rules introduced by right clauses (or the left rules introduced by left clauses) as definitions of their heads; the corresponding left rules (respectively right rules) arise by what Schroeder-Heister calls *definitional reflection* [13, p. 146]. Girard describes it [4, Section 1.2] as

> ... basically something like Clark's completion but less naïve: in some sense we want to say that if p has been introduced as head of the only clauses $C_1, \ldots, C_n \ldots$ then this must be taken as the definition of a new logical primitive (a generalised connective if one prefers), and that the elimination rules for p are basically the negation by failure rules. In this it is absolutely essential that we know that no other clause ending with p is there (i.e. the notion of *end* of the list of clauses that is essential in small details like failure).

Remark 2. Linearity is essential for this: we can, for example, use it to define a proposition p such that p is equivalent to $p^{\perp}$ [4, Section 2.1], which is harmless in linear logic but disastrous in classical or intuitionistic logic. This motivates the restriction on exponentials.

We now have

Proposition 1. *Linear logic, together with the rules introduced by a definition, satisfies cut elimination.*

Proof. For the case where we have right clauses only, we can use the proof in [13] (this theorem was announced, without proof, in [4]). A symmetry argument will, of course, give us the case of left clauses only, but for the mixed case we need

to encode left clauses as right clauses. Replace each left clause $\dfrac{T \vdash}{h(t_1,\ldots,t_n) \vdash}$ df with the pair of right clauses $\dfrac{\vdash T^{\perp}}{\vdash \tilde{h}(t_1,\ldots,t_n)}$ df and $\dfrac{\vdash \tilde{h}(t_1,\ldots,t_n)^{\perp}}{\vdash h(t_1,\ldots,t_n)}$ df, where $\tilde{h}$ is a new relational symbol; it is easy to check that the new system, which now consists only of right clauses, is equivalent to the stated rules. □

1 The Basic Representation

We will be comparing GOLOG (as defined in [6]) with a linear logic programming language, in such a way that Levesque *et al.*'s three-term relation $\mathrm{Do}(\delta, s, s')$ will correspond to the provability, in linear logic, of a sequent $s^{\otimes}, \overline{\delta} \vdash s'^{\otimes}$, for suitable formulae $s^{\otimes}$, $s'^{\otimes}$ and $\overline{\delta}$ corresponding to s, s' and δ respectively. There are, then, two issues: how we define the correspondences between the formulae, and how we formulate the correspondence.

The latter issue is rather easier to state: we can regard both terms of the comparison as labelled transition systems, the states s and s' (or their translations into linear logic) being the nodes, and the actions δ (respectively *their* translations) being the labelled edges. So we will want to show that this correspondence is a bisimulation.

1.1 Situations and Fluents

The question of the translation of the various terms is a little more delicate. Levesque *et al.* start with an informally described notion of situation, and then suppose that there are deterministic basic actions whose effects on situation can be determined using the machinery of the situation calculus [6, Sections 2.1–2.3]. Although this is all rather loosely specified, what is important about their approach seems to be the following:

1. States determine truth values of fluents; i.e. fluents can be regarded as predicates on the collection of states.

2. There are – for this form of the theory – no state constraints.

Remark 3. We use the notation of Levesque *et al.*, according to which fluents have a situation argument (and maybe others). Other authors – e.g. [7] – talk of fluents without a situation argument place; these correspond to what Levesque *et al.* call *pseudo-fluents*. The advantage of Levesque *et al.*'s, and our, approach is that it makes the type structure of the language more visible: fluents are functions from situations to propositions, and likewise actions are functions from situations to situations. This is mathematically illuminating and fits with an approach to language in the style of Montague: see [11] and [15, pp. 31ff.].

By contrast, the approach in which fluents (and actions) are atoms means that they can only be assigned relatively unstructured sorts, and that one then has to define auxiliary relations (holds, result, and so on) in order to recover the lost functional role of fluents and actions; this makes for a less transparent syntax.

Remark 4. There are a number of much debated issues here, to do with whether

1. situations are individuated by the propositions true and false in them, or by their causal history, and

2. we are considering the set of all (non-contradictory) situations, or merely the ones accessible from the initial situation.

Surprisingly little hangs on these: the fact that we will only be considering our representations of states and actions up to bisimulation means, in effect, that we will never have to make up our minds.

1.1.1 A First Attempt

We have already defined – in [16, 17] – a representation of situations and actions as linear logic terms. In this representation, situations are linear tensor products of atomic facts; this was adequate for the purposes of [16, 17], but we face difficulties if we try to extend this to GOLOG. The problem is this: in GOLOG, we want to test the truth-value of a given fluent in a given situation. According to the above account, fluents should be predicates of situations, and indeed it is easy enough, given a fluent – ϕ, say – to define a predicate of situations $\widetilde{\phi}$ such that, for any situation s, ϕ is true of s iff the sequent $\vdash \widetilde{\phi}(s^{\otimes})$ is valid. We will give such a definition in Section 1.3.

The representation of fluents, then, is not in itself problematic: the real difficulty lies elsewhere. Given a fluent ϕ, Levesque *et al.* define a *test action*, $\phi?$, such that $\mathrm{Do}(\phi?, s, s')$ succeeds iff $s = s'$ and ϕ is true in s. It is thus tempting to represent Levesque *et al.*'s test actions by terms such as $\forall X.(X \otimes \widetilde{\phi}(X)) \multimap X$, where X is a suitable variable. If ϕ is true in a situation s, then we certainly have a corresponding valid proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{s^{\otimes} \vdash s^{\otimes}} \quad \vdots}{\vdash \widetilde{\phi}(s^{\otimes})}}{s^{\otimes} \vdash s^{\otimes} \otimes \widetilde{\phi}(s^{\otimes})} \quad \dfrac{}{s^{\otimes} \vdash s^{\otimes}}}{s^{\otimes}, (s^{\otimes} \otimes \widetilde{\phi}(s^{\otimes})) \multimap s^{\otimes} \vdash s^{\otimes}}}{s^{\otimes}, \forall X.(X \otimes \widetilde{\phi}(X)) \multimap X \vdash s^{\otimes}}$$

with an appropriate proof of $\vdash \widetilde{\phi}(s^{\otimes})$.

However, we also get proofs where we do not want them. We can, that is, construct proofs such as the following:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{s^{\otimes} \vdash s^{\otimes}} \quad \vdots}{\vdash \widetilde{\phi}(s^{\otimes})}}{s^{\otimes}, (s^{\otimes} \otimes \widetilde{\phi}(s^{\otimes})) \multimap s^{\otimes} \vdash s^{\otimes}} \quad \dfrac{}{s'^{\otimes} \vdash s'^{\otimes}}}{s^{\otimes}, s'^{\otimes}, (s^{\otimes} \otimes \widetilde{\phi}(s^{\otimes})) \multimap s^{\otimes} \vdash s^{\otimes} \otimes s'^{\otimes}}}{s^{\otimes} \otimes s'^{\otimes}, (s^{\otimes} \otimes \widetilde{\phi}(s^{\otimes})) \multimap s^{\otimes} \vdash s^{\otimes} \otimes s'^{\otimes}}}{s^{\otimes} \otimes s'^{\otimes}, \forall X.(X \otimes \widetilde{\phi}(X)) \multimap X \vdash s^{\otimes} \otimes s'^{\otimes}}$$

This can lead to unintended proofs:

Example 1. Suppose that there are two swans, henry and bruce, and that henry is white whereas bruce is black. Let ϕ be the fluent corresponding to the proposition 'All swans are white'; then ϕ is true of the situation consisting only of 'henry is white', but false of the situation consisting of 'henry is white' and 'bruce is black'. In the linear logic representation, this means that $\vdash \widetilde{\phi}(\mathsf{white}(\mathsf{henry}))$ is valid, whereas $\vdash \widetilde{\phi}(\mathsf{white}(\mathsf{henry}) \otimes \mathsf{black}(\mathsf{bruce}))$ is not valid; the *predicate* part of the translation, then, behaves as we ought.

However, we have a proof of

$$\mathsf{white}(\mathsf{henry}) \otimes \mathsf{black}(\mathsf{bruce}), \forall X.(X \otimes \widetilde{\phi}(X)) \multimap X \vdash \mathsf{white}(\mathsf{henry}) \otimes \mathsf{black}(\mathsf{bruce})$$

constructed as above – $s^{\otimes}$ is $\mathsf{white}(\mathsf{henry})$ and $s''^{\otimes}$ is $\mathsf{black}(\mathsf{bruce})$ – even though the corresponding test action ought not to succeed on the situation in question.

More generally, we cannot, however we translate test actions into linear logic, correctly represent them in the style of [16, 17]; for it is characteristic of that representation that, if $s^{\otimes}, \overline{\delta} \vdash s'^{\otimes}$ is valid (where $s^{\otimes}$ and $s'^{\otimes}$ are representations of situations, and $\overline{\delta}$ is a representation of an action), then so too is $s^{\otimes} \otimes s''^{\otimes}, \overline{\delta} \vdash s'^{\otimes} \otimes s''^{\otimes}$. Consequently, if we could represent test actions in this way, then if a test action succeeded on a situation s, it ought also to succeed on all situations $s \otimes s''$; but this allows too many proofs to succeed, as the example shows.

The problem here lies, not in the translation of test actions, but in the use of $\otimes$ as a situation-forming operator. This use allows situations to be broken up into sub-situations, which are then processed independently. This is perfectly adequate for the actions that we were considering in [16, 17], and it gave a certain degree of asynchrony that could be computationally useful. However, as soon as we add test actions to the calculus, we need more synchronisation than we can have with $\otimes$ as a situation-forming operator. We must, then, define an alternative situation-forming operator, one which enforces the synchronisation that we need.

Remark 5. This is a theme that is, in another guise, quite common in the literature: it is well known that the situation calculus requires an implausibly large degree of synchronisation – indeed, the event calculus was defined in order to provide a less synchronous way of reasoning about actions [5]. However, the above example shows that we need some degree of synchronisation in order to apply tests to situations; it would be interesting to see if we could combine valid tests with some degree of asynchrony. Something in the style of [12] may well be possible.

1.1.2 The Situation-Forming Operators

We would like, then, to have situation-forming operators that would allow us to permute components of situations, but would not allow us to decompose them. We use Girard's Fixpoint Theorem to define an operator $\sigma(\cdot)$ as follows:

Definition 3.

$$\frac{\vdash \sigma(X \otimes (Y \otimes Z))}{\vdash \sigma((X \otimes Y) \otimes Z)}\,\mathsf{df} \qquad \frac{\vdash \sigma(Y \otimes X)}{\vdash \sigma(X \otimes Y)}\,\mathsf{df} \qquad \frac{\vdash \sigma(X)}{\vdash \sigma(X \otimes \mathbf{1})}\,\mathsf{df}$$

We now have:

Lemma 1. *$\sigma(X) \vdash \sigma(Y)$ iff X and Y are linear tensor products of terms, equal up to permutation and units.*

Proof. Suppose we have such a proof, By cut elimination, the only rules which are ever applicable on the proof tree are the right and left rules for $\sigma(\cdot)$ and the linear logic axioms; the result then follows by induction over the proof tree.

Conversely, if the two terms are equal up to permutation and units, it is easy to concoct such a proof. □

1.1.3 Permutabilities

We need the following easy permutability result:

Lemma 2. *If $\Gamma, \sigma(A_1), \ldots, \sigma(A_i) \vdash \sigma(B_1), \ldots, \sigma(B_j), \Delta$ is valid, then there is a proof in which all applications of the σ-rules occur after (i.e. above) all applications of other rules.*

Proof. Suppose that we have a proof. Notice that the σs can occur as principal formulae only in the σ-rules themselves, or in axioms; the axioms, of course, can only occur at the top of the tree.

Take the proof, and remove all instances of the σ-rules from it as follows, starting from the bottom. If we have a right σ-rule, then it is of the form

$$\dfrac{\begin{array}{c}\Pi \\ \vdots \\ \Gamma \vdash \sigma(B'), \Delta\end{array}}{\Gamma \vdash \sigma(B), \Delta}$$

Replace all instances of $\sigma(B')$ in Π with $\sigma(B)$, and delete the σ rule. If we have a left σ-rule, then it must be of the form

$$\dfrac{\begin{array}{c}\Pi \\ \vdots \\ \Gamma, \sigma(A') \vdash \Delta\end{array} \qquad \begin{array}{c}\vdots \\ \Gamma, \sigma(A'') \vdash \delta\end{array} \qquad \begin{array}{c}\vdots \\ \cdots\end{array}}{\Gamma, \sigma(A) \vdash \Delta}$$

Select one of the branches (e.g. the left one), delete the σ-rule, and replace all instances of $\sigma(A')$ in Π with $\sigma(A)$.

After we have done this, we will have a tree in which all rules are valid except for those in which one of the σs is principal; since we have removed all of the σ rules, the only such rules remaining are the axioms. We thus have a tree which is a valid proof tree except that it has some leaves of the form $\sigma(A) \vdash \sigma(B)$. However, A is a replacement for some A', and B is a replacement for something, which must be A' because the proof was valid before we modified it. Furthermore, we can check inductively that we always replaced terms with equivalent ones, up to permutation and units; thus, A and B must be equivalent up to permutation and units. Consequently there is a proof of $\sigma(A) \vdash \sigma(B)$ which involves only the σ rules; we graft suitable such proofs onto such leaves as are necessary, and we have a valid proof of the required form. □

We also need the following rather trivial lemma:

Lemma 3. *The only valid sequents $\Gamma \vdash \Delta$, where Γ and Δ are multisets of propositions of the form $\sigma(\cdot)$, are $\sigma(A) \vdash \sigma(B)$, where A and B are equal up to permutation and units.*

Proof. The only rules that one can apply to such sequents are the left and right rules for σ, and the axioms. The former leave unchanged the cardinality of the multisets on either side of the sequent, whereas we can only apply the axioms if there is exactly one formula to the left and right. □

1.2 Representing Situations

We shall, then, represent situations by linear logic terms of the form $\sigma(x \otimes y \otimes \ldots)$. For a GOLOG situation s, we let $s^{\otimes}$ be the tensor product of the atoms which are true in s; so we will represent s by the linear logic term $\sigma(s^{\otimes})$.

Example 2. The "situation consisting of 'henry is white' and 'bruce is black'" of Example 1 will be represented as $\sigma(\mathsf{white}(\mathsf{henry}) \otimes \mathsf{black}(\mathsf{bruce}))$.

1.2.1 Basic Actions

We will also define a class of basic actions; this is rather smaller than Levesque *et al.*'s class of basic actions, but it is all we need. As it turns out, we can define all of Levesque *et al.*'s rather loosely defined class in terms of ours together with their action connectives.

Definition 4.

$$\frac{\forall Z.\, \sigma(X \otimes Z) \multimap \sigma(Y \otimes Z) \vdash}{\alpha(X, Y) \vdash} \mathsf{df}$$

We have the following:

Proposition 2. *$\sigma(A), \alpha(X, Y) \vdash \sigma(B)$ is valid iff $A \otimes Y$ is equivalent to $B \otimes X$, up to permutation and units.*

Proof. If is clear. Suppose that $A \otimes Y$ is equivalent to $B \otimes X$; then there is a W such that A is equivalent to $X \otimes W$ and B is equivalent to $Y \otimes W$. We use the following proof.

$$\dfrac{\dfrac{\dfrac{\begin{array}{cc}\Pi_1 & \Pi_2 \\ \vdots & \vdots \\ \sigma(A) \vdash \sigma(X \otimes W) & \sigma(Y \otimes W) \vdash \sigma(B)\end{array}}{\sigma(A),\, \sigma(X \otimes W) \multimap \sigma(Y \otimes W) \;\vdash\; \sigma(B)}}{\sigma(A),\, \forall Z.\, \sigma(X \otimes Z) \multimap \sigma(Y \otimes Z) \;\vdash\; \sigma(B)}}{\sigma(A),\, \alpha(X, Y) \;\vdash\; \sigma(B)} *$$

using the left rule for α at $*$, and where Π_1 and Π_2 are given by Lemma 1.

For the only if direction, suppose that we have a proof of the sequent in question. By Lemma 2 we can assume, without loss of generality, that the proof starts with an application of the left rule for α; we are therefore reduced to proving the condition on A, B, X, and Y given the validity of $\sigma(A), \forall Z.\sigma(X \otimes Z) \multimap \sigma(Y \otimes Z) \vdash \sigma(B)$. Again without loss of generality we can assume that

we now have an application of the left $\forall$ rule. For some W, we must have the validity of $\sigma(A), \sigma(X \otimes W) \multimap \sigma(Y \otimes W) \vdash \sigma(B)$; again we can assume that we first apply the left $\multimap$ rule. The only distribution of resources that leads to a proof is

$$\frac{\begin{array}{cc}\vdots & \vdots \\ \sigma(A) \vdash \sigma(X \otimes W) & \sigma(Y \otimes W) \vdash \sigma(B)\end{array}}{\sigma(A),\ \sigma(X \otimes W) \multimap \sigma(Y \otimes W) \ \vdash\ \sigma(\sigma(B))}$$

and, by Lemma 1, we get the result. □

Lemma 4. *There are no proofs of*

$$\Gamma,\ \alpha(X, Y) \ \vdash\ \Delta,$$

where Γ *and* Δ *are multisets of propositions of the form* $\sigma(\cdot)$*, if either* Γ *or* Δ *is empty.*

Proof. Because of the permutabilities, we can assume that any proof starts:

$$\dfrac{\dfrac{\dfrac{\begin{array}{cc}\Pi_1 & \Pi_2 \\ \vdots & \vdots \\ \Gamma \vdash \sigma(X \otimes W), \Delta & \Gamma', \sigma(Y \otimes W) \vdash \Delta'\end{array}}{\Gamma, \Gamma',\ \sigma(X \otimes W) \multimap \sigma(Y \otimes W) \ \vdash\ \Delta, \Delta'}}{\Gamma, \Gamma',\ \forall Z.\, \sigma(X \otimes Z) \multimap \sigma(Y \otimes Z) \ \vdash\ \Delta, \Delta'}}{\Gamma, \Gamma',\ \alpha(X, Y) \ \vdash\ \Delta, \Delta'}$$

Π_1 cannot be valid unless Γ is non-empty, and Π_2 cannot be valid unless Δ' is non-empty. □

Example 3. The basic actions described in [6, Section 2.2] can be fitted into this framework. For example, the pickup(x) action (by which the agent picks up an object x) replaces notHolding(x) by holding(x), and can thus be represented as $\alpha(\mathsf{notHolding}(x), \mathsf{holding}(x))$. Actions quite often have preconditions: thus the preconditions of the pickup action are

1. that the agent is not holding the object initially,
2. that the agent is next to the object, and
3. that the object is not heavy.

These preconditions can be accomodated by working with a sequence of suitable test actions followed by the appropriate α; both test actions and sequence will be described in Section 2.

One should also notice that in many cases the preconditions are not necessary in the linear logic formalism. Thus the action $\alpha(\mathsf{notHolding}(x), \mathsf{holding}(x))$ will simply not succeed in any situation which does not have notHolding(x) as a component; using Proposition 2 it is easy to show that there can be no proof of any sequent

$$\sigma(A),\ \alpha(\mathsf{notHolding}(x), \mathsf{holding}(x)) \ \vdash\ \sigma(B)$$

unless A contains $\mathsf{notHolding}(x)$ as a component. So one does not need to give precondition 1 explicitly. One could, similarly, absorb the third precondition into the α term, writing the action $\alpha(\mathsf{notHolding}(x) \otimes \mathsf{notHeavy}(x), \mathsf{Holding}(x) \otimes \mathsf{notHeavy}(x))$. One might, however, have qualms about such an action description, although it is a relatively common idiom in STRIPS. (Such things as the $\mathsf{notHeavy}(x)$ fluent, which figure unchanged on both input and output of the action description, are sometimes described as *prevailing* fluents.)

There are actions with effects which depend on the initial state of the system. Thus Levesque *et al.* have an drop action, which breaks fragile things:

$$\mathsf{poss}(\mathsf{drop}(x), s) \wedge \mathsf{fragile}(x, s) \rightarrow \mathsf{broken}(x, \mathsf{do}(\mathsf{drop}(x), s))$$

(i.e. if, in a situation s, it is possible to drop x, and if x is fragile in s, then x is broken in the situation which is the result of dropping x in s). The result of dropping an object thus depends on whether it is fragile or not. This sort of dependency can be handled by a choice between two actions, one of which has the precondition of fragile and which makes the object broken, and the other of which has the precondition of $\neg$fragile and which leaves the object intact. Choice will, again, be described in Section 2.

Finally, some actions have logically more complex effects: thus there is an explode action which has the effect of breaking every object near to the exploding object. Such effects can, again, be described in terms of a while construct, and this can, in turn, be described in terms of the usual composite action constructs of Section 2 – indeed, Levesque *et al.* have a very similar, but less destructive, action, which (in the blocks world) puts all the blocks away into the box: [6, p. 7].

Thus, although our set of basic actions is rather more austere than Levesque *et al*'s, we can describe all of their basic actions in terms of composites of our primitives.

1.3 Representing Fluents

Levesque *et al.* assume that we are given fluents, and corresponding test actions, without giving any syntax for these. I give here a syntax, and then show how to define corresponding predicates in linear logic.

1.3.1 The Fluents

We assume that we are given some finite collection of individuals, for example narcissus, block_5, and so on. We assume that all of the individuals have names. We assume that we are given some class of atoms, for example $\mathsf{loves}(\mathsf{narcissus}, \mathsf{narcissus})$, $\mathsf{on}(\mathsf{table}, \mathsf{book})$, $\mathsf{on}(\mathsf{floor}, X)$, and so on. The argument places of these atoms may be occupied by individuals or by free variables. *Ground* atoms are those in which there are no free variables.

Definition 5.
- If α is a (not necessarily ground) atom, then ϕ_α is a *schematic test*;
- If ϕ is a schematic test, then $\forall x.\phi$ and $\exists x.\phi$, and $\neg\phi$ are schematic tests;
- If ϕ and ϕ' are schematic tests, then $\phi \vee \phi'$, $\phi \wedge \phi'$, and $\phi \rightarrow \phi'$ are schematic tests;

- A schematic test whose only free variable is the situation argument is a test.

We define truth conditions as follows:

Definition 6.
- Let α be a ground atom: the *elementary α-test*, ϕ_α, is a test. $\phi_\alpha(s)$ is true of a situation s iff α is a component of s;
- $\forall x.\phi(s)$ is true iff, for all ground substitutions for x, the resulting test is true;
- $\exists x.\phi(s)$ is true iff, for some substitution for x, the resulting test is true;
- boolean combinations of tests are evaluated in the usual way.

1.3.2 Representing Fluents in Linear Logic

We can now define the translations of our tests. Suppose first that $\alpha_1, \ldots, \alpha_n$ are all the individuals that there are. First define a predicate, $\mathcal{I}(\cdot)$, which will pick out our individuals:

Definition 7. Define the predicate $\mathcal{I}(\cdot)$ by the set of clauses:

$$\left\{ \frac{\vdash \mathbf{1}}{\vdash \mathcal{I}(\alpha_i)}\,\mathsf{df} \right\}_{i=1}^{n}$$

Remark 6. We have assumed that it is practical to simply list all of the individuals that there are, and also that individuals have unique names. This is a little naive: one would like to have more syntax, in order to deal with perfectly good first-order terms like 'the Moor's last sigh' and 'the third man'. In order to deal with intensionality, one would also like to have a decidable equality on such terms. Given such a syntax – say along the lines of [11] – a corresponding $\mathcal{I}(\cdot)$ predicate could be defined.

We define the translation, $\widetilde{\phi}$, of a fluent, ϕ, as follows. We define linear logic predicates corresponding to elementary tests and the negations of elementary tests. In order to translate a fluent of arbitrary complexity, we first convert it to an equivalent form in which negations only apply to atoms, and in which the only connectives are $\wedge$, $\vee$, and the quantifiers, and then define a translation using the translations of atoms, negated atoms, and the linear logic quantifiers.

Example 4. Continuing Example 1, we can define tests such as $\phi_{\mathsf{white(bruce)}}$, $\phi_{\mathsf{white(henry)}}$, $\phi_{\mathsf{black(bruce)}}$, and so on. Then $\phi_{\mathsf{white(bruce)}}$ is true of a situation in which bruce is white and false of any situation in which bruce is not white. We are assuming that all situations are non-contradictory; they will not, for example, simultaneously contain white(bruce) and black(bruce). We are also relying – as do Levesque *et al.* – on a rather informal understanding of what situations are: in particular, the idea of a "component of a situation" remains unexplained. We will be able to be more precise when we deal with the linear logic translations of these things.

So, first the base cases:

Definition 8.

- If α is a ground atom, we define the predicate $\widetilde{\phi_\alpha}$ by the right clause:

$$\frac{\vdash \mathbf{1}}{\vdash \widetilde{\phi_\alpha}(X \otimes \alpha \otimes Y)}\,\mathsf{df}$$

- If α is a ground atom, we define the predicate $\widetilde{\neg\phi_\alpha}$ by the left clause:

$$\frac{\bot\vdash}{\widetilde{\neg\phi_\alpha}(X \otimes \alpha \otimes Y)\vdash}\,\mathsf{df}$$

Then the recursive cases:

Definition 9.

- $\widetilde{\phi \wedge \psi} = \widetilde{\phi}\&\widetilde{\psi}$,
- $\widetilde{\phi \vee \psi} = \widetilde{\phi} \oplus \widetilde{\psi}$,
- $\widetilde{\forall x.\phi(x)} = \forall x.\mathcal{I}(x) \multimap \widetilde{\phi(x)}$,
- $\widetilde{\exists x.\phi(x)} = \exists x.\mathcal{I}(x) \otimes \widetilde{\phi(x)}$.

We now have

Proposition 3. *ϕ is true of a situation s iff there is a proof of $\vdash \widetilde{\phi}(s^{\otimes})$.*

Proof. We prove this by the usual recursion. Firstly, if the test is of the form ϕ_α, for an atom α, we can use cut elimination to show that the only proofs of $\vdash \widetilde{\phi_\alpha}(s)$ use a right rule for α, and will only succeed if s contains α. This is clearly correct.

Secondly, if the test is of the form $\neg\phi_\alpha$, for some atom α, then it is clear that, if s does not contain α, then there is a proof of $\widetilde{\neg\phi_\alpha}(s)$ – because the antecedent of the right rule is an empty set of sequents – whereas if s contains α, we can only apply the right rule, and we get one or more sequents of the form $\vdash\bot$, which are unprovable.

That completes the base cases. We handle the recursive cases as follows. We want there to be a proof of $\vdash \widetilde{(\phi \wedge \psi)}(s)$ iff there is a proof of $\vdash \widetilde{\phi}(s)$ and a proof of $\vdash \widetilde{\psi}(s)$; but $\widetilde{(\phi \wedge \psi)}(s)$ is just $\widetilde{\phi}(s)\&\widetilde{\psi}(s)$ and, by cut elimination, proofs of it can be put into the form:

$$\frac{\begin{array}{cc}\Pi_1 & \Pi_2\\ \vdots & \vdots\\ \vdash \widetilde{\phi}(s) & \vdash \widetilde{\psi}(s)\end{array}}{\vdash \widetilde{\phi}(s)\&\widetilde{\psi}(s)}$$

and the result is immediate. The proof for $\widetilde{\phi \vee \psi}$ is much the same.

For $\widetilde{\forall x.\phi(x)}$, we want to prove that there is a proof of $\vdash \widetilde{\forall x.\phi(x)}(s)$ iff, for all individuals α_i, there is a proof of $\widetilde{\phi(\alpha_i)}(s)$. Now $\widetilde{\forall x.\phi(x)}(s)$ is $\forall x.\mathcal{I}(x) \multimap$

$\widetilde{\phi(x)}(s)$, and, by cut elimination, proofs can be put into the form (where $\alpha_1 \ldots \alpha_n$ are all of the individuals that there are):

$$\dfrac{\dfrac{\dfrac{\dfrac{\Pi_1}{\vdots}}{\vdash \widetilde{\phi(\alpha_1)}(s)} \quad \cdots \quad \dfrac{\dfrac{\Pi_n}{\vdots}}{\vdash \widetilde{\phi(\alpha_n)}(s)}}{\mathbf{1} \vdash \widetilde{\phi(\alpha_1)}(s) \quad \cdots \quad \mathbf{1} \vdash \widetilde{\phi(\alpha_n)}(s)}\,\dagger}{\dfrac{\dfrac{\mathcal{I}(x) \vdash \widetilde{\phi(x)}(s)}{\vdash \mathcal{I}(x) \multimap \widetilde{\phi(x)}(s)}}{\vdash \forall x.\mathcal{I}(x) \multimap \widetilde{\phi(x)}(s)}\,\text{new } x}$$

Here $\dagger$ is an application of the left rule for $\mathcal{I}(\cdot)$. The only tricky point is $\dagger$; one might think that one could also apply here a right rule for $\phi(x)$, but this would not be valid – x here is a free variable, and the form of right rules forbid instantiating it. Given that proofs are of this form, the result is immediate.

The case for $\widetilde{\exists x.\phi(x)}(s)$ is quite similar. □

Remark 7. The above result entails, for example, that $\vdash \phi \wedge \widetilde{(\psi \vee \theta)}(s)$ is provable iff $\vdash \widetilde{(\phi \wedge \psi) \vee (\phi \wedge \theta)}(s)$ is provable, and it would be tempting to regard this as an instance of & distributing over $\oplus$. However, & doesn't distribute over $\oplus$ – $A\&(B \oplus C) \vdash (A\&B) \oplus (A\&C)$ is not provable – and the above biconditional is only admissible, not derivable.

2 Translating Composite Actions

We have, so far, defined basic actions and situations in our language. We can now define the translations of composite actions, following [6, Section 3.1].

2.1 Bisimulation

The verification that the translation has the correct properties will be an inductive verification of the usual sort. The main property to be verified will be bisimulation. Since we have to carry out an inductive proof, the property that we have to verify is rather more elaborate than we might suspect: the tricky point will be the distribution of resources in proofs, and the fourth clause in the definition of bisimulation is necessary in order to guarantee that proofs of composite actions distribute their resources to subproofs in the correct way.

Definition 10. A linear logic term $\overline{\delta}$ *bisimulates* an action δ if

1. for situations s_1 and s_2, if $\mathrm{Do}(s_1, \delta, s_2)$, then there is a proof of $\sigma({s_1}^{\otimes}), \overline{\delta} \vdash \sigma({s_2}^{\otimes})$;
2. for a situation s_1, if there is a proof of $\sigma({s_1}^{\otimes})\overline{\delta} \vdash \sigma(B)$, then there is a situation s_2 with $\mathrm{Do}(s_1, \delta, s_2)$, and (up to permutation and units) $B = {s_2}^{\otimes}$;
3. for a situation s_2, if there is a proof of $\sigma(\sigma(A)), \overline{\delta} \vdash \sigma(\sigma {s_2}^{\otimes})$, then there is a situation s_1 with $\mathrm{Do}(s_1, \delta, s_2)$, and (up to permutation and units) $A = {s_1}^{\otimes}$;

4. Any valid proof of $\Gamma, \overline{\delta} \vdash \Delta$, where Γ and Δ are multisets of propositions of the form $\sigma(\cdot)$, must have Γ and Δ non-empty.

So far we have only defined basic actions – in Section 1.2.1 – so we should verify that they satisfy bisimulation. We are not, of course, interested in just any action $\alpha(X, Y)$; rather, we have a pre-existing set of GOLOG basic actions, and we pick $\alpha(X, Y)$s to match them. So we can assume that, for each of these $\alpha(X, Y)$s, there is a suitable GOLOG action – call it $a_{X,Y}$ – such that, for any s and s', $\mathrm{Do}(a_{X,Y}, s, s')$ iff $\sigma(s^{\otimes}), \alpha(X, Y) \vdash \sigma(s'^{\otimes})$. This gives us the first clause of the bisimulation definition for free; the other two also hold.

Lemma 5. *The basic actions $\alpha(X, Y)$ for which there are* GOLOG *counterparts $a_{X,Y}$ satisfy bisimulation.*

Proof. By the permutabilities, proofs of the sequent in question can be assumed to begin (writing Γ, etc., for the relevant multisets of $\sigma(\cdot)$s):

$$\dfrac{\dfrac{\begin{array}{cc}\Pi & \Pi' \\ \vdots & \vdots \\ \Gamma \vdash \sigma(X \otimes Z), \Delta & \Gamma', \sigma(Y \otimes Z) \vdash \Delta'\end{array}}{\Gamma, \Gamma', \sigma(X \otimes Z) \multimap \sigma(Y \otimes Z) \vdash \Delta, \Delta'}}{\Gamma, \Gamma', \forall Z. \sigma(X \otimes Z) \multimap \sigma(Y \otimes Z) \vdash \Delta, \Delta'}$$

Now suppose that $\Gamma \sqcup \Gamma' = \{\!|\sigma(s^{\otimes})|\!\}$. Since Π' is valid, Γ' must be empty; so we have $\Gamma = \{\!|\sigma(s^{\otimes})|\!\}$, and, since Π is valid, Δ must be empty. Since Π' is valid, finally, we must have $\Delta' = \{\!|\sigma(s'^{\otimes})|\!\}$, and we can use Prop 2 to show that s and s' must be appropriately related. This proves the second clause of the definition of bisimulation; the third clause is similar. For the fourth clause, we use Lemma 4. □

2.2 Null Actions

We will also need a null action, which does nothing: we define it by

Definition 11. Null $\stackrel{\mathsf{df}}{=}$ **1.**

This clearly bisimulates the GOLOG null action, which Levesque *et al.* omit to define, but which has the obvious semantics.

2.3 Test Actions

Levesque *et al.* define a test action as follows, where ϕ is a fluent and $\phi?$ is the corresponding test action:

$$\mathrm{Do}(\phi?, s, s') \stackrel{\mathsf{df}}{=} \phi(s) \wedge (s = s'). \tag{1}$$

We assume (as explained previously) that we have a suitable translation $\widetilde{\phi}$ of ϕ. We now define

Definition 12.

$$\overline{(\phi?)} \stackrel{\mathsf{df}}{=} \forall X. (\sigma(X) \otimes \widetilde{\phi}(X)) \multimap \sigma(X)$$

Now we have

Lemma 6. *$\overline{\phi?}$ bisimulates $\phi?$.*

Proof. Suppose first that we have $\mathrm{Do}(\phi?, s, s')$; we must then have $s = s'$and $\phi(s)$. By Prop 3, we have a proof Π of $\vdash \widetilde{\phi}(s^{\otimes})$. So we have a proof of $\sigma(s^{\otimes}), \overline{\phi?} \vdash \sigma(s^{\otimes})$, namely

$$\dfrac{\dfrac{\dfrac{\vdots \quad \Pi \atop \vdots \quad \vdots}{\sigma(s^{\otimes}) \vdash \sigma(s^{\otimes}) \qquad \vdash \widetilde{\phi}(s^{\otimes})} \qquad \vdots}{\sigma(s^{\otimes}) \vdash \sigma(s^{\otimes}) \otimes \widetilde{\phi}(s^{\otimes}) \qquad \sigma(s^{\otimes}) \vdash \sigma(s^{\otimes})}}{\sigma(s^{\otimes}), (\sigma(s^{\otimes}) \otimes \widetilde{\phi}(s^{\otimes})) \multimap \sigma(s^{\otimes}) \vdash \sigma(s^{\otimes})}}{\sigma(s^{\otimes}), \forall X.(\sigma(X) \otimes \widetilde{\phi}(X)) \multimap \sigma(X) \vdash \sigma(s^{\otimes})}$$

We now consider the second and third clauses of the definition of bisimulation. By the permutabilities, we can assume that proofs of

$$\Gamma, \forall X.(\sigma(X) \otimes \widetilde{\phi}(X)) \multimap \sigma(X) \vdash \Delta$$

are of the form

$$\dfrac{\dfrac{\dfrac{\Pi_1 \quad \Pi_2}{\vdots \quad \vdots}}{\Gamma \vdash \sigma(X), \Delta \qquad \Gamma' \vdash \widetilde{\phi}(X)), \Delta'} \qquad \Pi_3 \atop \vdots}{\Gamma, \Gamma' \vdash \sigma(X) \otimes \widetilde{\phi}(X))\Delta, \Delta' \qquad \Gamma'', \sigma(X) \vdash \Delta''}}{\Gamma, \Gamma', \Gamma'', (\sigma(X) \otimes \widetilde{\phi}(X)) \multimap \sigma(X) \vdash \Delta, \Delta', \Delta''}}{\Gamma, \Gamma', \Gamma'', \forall X.(\sigma(X) \otimes \widetilde{\phi}(X)) \multimap \sigma(X) \vdash \Delta, \Delta', \Delta''}$$

Now suppose that $\Gamma \sqcup \Gamma' \sqcup \Gamma'' = \lbag \sigma(s^{\otimes}) \rbag$. Because Π_1 is valid, we must have $\Gamma = \lbag \sigma(X) \rbag$ and $\Delta = \emptyset$. Because Π_3 is valid, we must have $\Gamma'' = \emptyset$ and $\Delta'' = \lbag \sigma(X) \rbag$. So we must have $\Gamma' = \Gamma'' = \emptyset$, and $X = s^{\otimes}$, because $\Gamma \sqcup \Gamma' \sqcup \Gamma''$ is the singleton $\lbag \sigma(s^{\otimes}) \rbag$. Furthermore, $\Delta \sqcup \Delta' \sqcup \Delta''$ is, by assumption, a singleton, so we have $\Delta = \Delta' = \emptyset$ and $\Delta'' = \lbag \sigma(s^{\otimes}) \rbag$. Thus, Π_2 is a proof of the sequent $\vdash \widetilde{\phi}(s^{\otimes})$; by Proposition 3 we must have $\phi(s)$. Consequently we have $\mathrm{Do}(\phi?, s, s)$. Thus the second clause is satisfied. The third clause is exactly similar. For the fourth clause, we argue that, since Π_1 is valid, Γ must be non-empty; similarly, since Π_3 is valid, Δ'' must be non-empty. □

2.4 Sequence

The semantics of Levesque *et al.*'s sequence operator ; is defined by follows:

$$\mathrm{Do}((\delta_1; \delta_2), s, s') \stackrel{\mathsf{df}}{=} \exists s^* \mathrm{Do}(\delta_1, s, s^*) \wedge \mathrm{Do}(\delta_2, s^*, s') \tag{2}$$

We define our translation of ; thus:

Definition 13.

$$\overline{(\delta_1; \delta_2)} \stackrel{\mathsf{df}}{=} \forall Z.(\overline{(\delta_1)} \otimes \sigma(Z)^{\perp}) \parr (\sigma(Z) \otimes \overline{(\delta_2)})$$

We now have:

Lemma 7. *If $\overline{\delta_1}$ and $\overline{\delta_2}$ bisimulate δ_1 and δ_2 then $\overline{(\delta_1;\delta_2)}$ bisimulates $\delta_1;\delta_2$.*

Proof. Suppose first that, for situations s_1 and s_2, $\mathrm{Do}((\delta_1;\delta_2), s_1, s_2)$. By the the semantics of GOLOG, we have a situation s_3 such that $\mathrm{Do}(\delta_1, s_1, s_3)$ and $\mathrm{Do}(\delta_2, s_3, s_2)$. By bisimulation for δ_1 and δ_2, we have proofs Π_1 and Π_2 of $\sigma(s_1{}^{\otimes}), \overline{\delta_1} \vdash \sigma(s_3{}^{\otimes})$ and $\sigma(s_3{}^{\otimes}), \overline{\delta_2} \vdash \sigma(s'{}^{\otimes})$. We thus get a proof of $\sigma(s^{\otimes}), \overline{(\delta_1;\delta_2)} \vdash \sigma(s'{}^{\otimes})$, viz:

$$\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\Pi_1\\ \vdots\\ \sigma(s_1{}^{\otimes}), \overline{\delta_1} \vdash \sigma(s_3{}^{\otimes})\end{array}}{\sigma(s_1{}^{\otimes}), \overline{\delta_1}, \sigma(s_3{}^{\otimes})^{\perp} \vdash}}{\sigma(s_1{}^{\otimes}), \overline{\delta_1} \otimes \sigma(s_3{}^{\otimes})^{\perp} \vdash} \qquad \begin{array}{c}\Pi_2\\ \vdots\\ \sigma(s_3{}^{\otimes}) \otimes \overline{\delta_2} \vdash \sigma(s_2{}^{\otimes})\end{array}}{\sigma(s_1{}^{\otimes}), (\overline{\delta_1} \otimes \sigma(s_3{}^{\otimes})^{\perp}) \parr (\sigma(s_3{}^{\otimes}) \otimes \overline{\delta_2}) \vdash \sigma(s_2{}^{\otimes})}\,\dagger}{\sigma(s_1{}^{\otimes}), \forall Z.\, (\overline{\delta_1} \otimes \sigma(Z)^{\perp}) \parr (\sigma(Z) \otimes \overline{\delta_2}) \vdash \sigma(s_2{}^{\otimes})}$$

This gives the first clause of the definition of bisimulation.

For the other clauses, we use the usual permutability results to argue that, wlog, proofs of sequents such as $\Gamma, \overline{(\delta_1;\delta_2)} \vdash \Delta$ are of the form:

$$\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\Pi_1\\ \vdots\\ \Gamma, \overline{\delta_1} \vdash \sigma(C), \Delta\end{array}}{\Gamma, \overline{\delta_1}, \sigma(C)^{\perp} \vdash \Delta}}{\Gamma, \overline{\delta_1} \otimes \sigma(C)^{\perp} \vdash \Delta} \qquad \begin{array}{c}\Pi_2\\ \vdots\\ \Gamma', \sigma(C), \overline{\delta_2} \vdash \Delta'\end{array}}{\Gamma, \Gamma', (\overline{\delta_1} \otimes \sigma(C)^{\perp}) \parr (\sigma(C) \otimes \overline{\delta_2}) \vdash \Delta, \Delta'}\,\dagger}{\Gamma, \Gamma', \forall Z.\, (\overline{\delta_1} \otimes \sigma(Z)^{\perp}) \parr (\sigma(Z) \otimes \overline{\delta_2}) \vdash \Delta, \Delta'}$$

Suppose now, for the third clause, that $\Gamma \sqcup \Gamma' = \lbag \sigma(s_1{}^{\otimes}) \rbag$ and that $\Delta \cup \Delta'$ is a singleton $\lbag \sigma(B) \rbag$. By bisimulation for $\overline{\delta_1}$, Γ must be non-empty, and must therefore be $\lbag \sigma(s_1{}^{\otimes}) \rbag$; Γ' must therefore be empty. By bisimulation for $\overline{\delta_2}$, we must have Δ' non-empty; because $\Delta \cup \Delta'$ is a singleton, Δ must be empty. Now by bisimulation for δ_1, there must be a situation s_3, with $C = s_3{}^{\otimes}$, such that $\mathrm{Do}(\delta_1, s_1, s_3)$. We can now apply bisimulation for δ_2 to show that there is a situation s_2, such that $\mathrm{Do}(\delta_2, s_3, s_2)$, and that $\Delta' = \lbag \sigma(s_2{}^{\otimes}) \rbag$. This proves the second clause of bisimulation.

Suppose now, for the third clause, that $\Delta \cup \Delta' = \lbag \sigma(s_2{}^{\otimes}) \rbag$ and that $\Gamma \cup \Gamma'$ is a singleton $\lbag \sigma(A) \rbag$. By bisimulation for δ_1, Γ must be non-empty, so consequently Γ' must be empty. By bisimulation for δ_2, Δ' must be non-empty, so Δ must be empty. We can now apply bisimulation for δ_2 to show that $C = s_3{}^{\otimes}$ for some s_3 with $\mathrm{Do}(\delta_2, s_3, s_2)$; consequently we can apply bisimulation for δ_1 to show that $\Gamma = \lbag \sigma(s_1{}^{\otimes}) \rbag$ for some s_1 with $\mathrm{Do}(\delta_1, s_1, s_3)$. This proves the second clause.

Finally, to prove the fourth clause, we argue that – for general Γ and $\Gamma' - \Gamma$ cannot be empty, by bisimulation for δ_1. So $\Gamma \sqcup \Gamma'$ cannot be empty. Similarly $\Delta \sqcup \Delta'$ cannot be empty. □

2.5 Nondeterministic Choice of Actions

Levesque *et al.*'s nondeterministic choice is defined as follows:

$$\mathrm{Do}((\delta_1|\delta_2), s, s') \stackrel{\mathrm{df}}{=} \mathrm{Do}(\delta_1, s, s') \vee \mathrm{Do}(\delta_2, s, s') \tag{3}$$

We define inductively our translation of $|$ as follows:

Definition 14.

$$\overline{(\delta_1|\delta_2)} \stackrel{\mathrm{df}}{=} \overline{\delta_1}\&\overline{\delta_2}$$

We now have:

Lemma 8. *If $\overline{\delta_1}$ and $\overline{\delta_2}$ bisimulate δ_1 and δ_2, then $\overline{(\delta_1|\delta_2)}$ bisimulates $\delta_1|\delta_2$.*

Proof. By the permutabilities, for any A and B, we can assume that proofs of $\Gamma, \overline{\delta_1|\delta_2} \vdash \Delta$ are of the form

$$\dfrac{\begin{array}{c}\Pi \\ \vdots \\ \Gamma, \overline{\delta_1} \vdash \Delta\end{array}}{\Gamma, \overline{\delta_1}\&\overline{\delta_2} \vdash \Delta}$$

or of the form

$$\dfrac{\begin{array}{c}\Pi \\ \vdots \\ \Gamma, \overline{\delta_2} \vdash \Delta\end{array}}{\Gamma, \overline{\delta_1}\&\overline{\delta_2} \vdash \Delta}$$

so facts about resource distribution, etc., for $\overline{(\delta_1|\delta_2)}$ are – in the first case – facts about resource distribution for $\overline{\delta_1}$, or – in the second case – facts about resource distribution for $\overline{\delta_2}$. This gives us the various clauses in the definition of bisimulation. □

2.6 Nondeterministic Choice of Arguments

Levesque *et al.*'s nondeterministic choice of action arguments is defined as follows:

$$\mathrm{Do}(\pi x.\delta(x), s, s') \stackrel{\mathrm{df}}{=} \exists x.\mathrm{Do}(\delta(x), s, s') \tag{4}$$

We define our translation as follows:

Definition 15.

$$\overline{\pi x.\delta(x)} \stackrel{\mathrm{df}}{=} \forall x.\overline{\delta(x)}$$

We now have:

Lemma 9. *Suppose that, for all instantiations $[t/x]$ of x in $\delta(x)$, $\overline{\delta[t/x]}$ bisimulates $\delta[t/x]$, then $\overline{\pi x.\delta(x)}$ bisimulates $\pi x.\delta(x)$.*

Proof. Suppose that we have $\sigma(A^{\otimes}), \overline{\pi x.\delta(x)} \vdash \sigma(B^{\otimes})$ for some A and B. Wlog, we can assume that proofs are of the following form:

$$\frac{\begin{array}{c}\Pi \\ \vdots \\ \sigma({s_1}^{\otimes}), \overline{\delta}[t/x] \vdash \sigma({s_2}^{\otimes})\end{array}}{\sigma({s_1}^{\otimes}), \forall x.\overline{\delta} \vdash \sigma({s_2}^{\otimes})}$$

for some Π.

Suppose first that we have $\mathrm{Do}(\pi x.\delta(x), s_1, s_2)$; by the GOLOG semantics, we must have $\mathrm{Do}(\delta(t), s_1, s_2)$ for some term t. By the assumption, then, we have a proof of $\sigma({s_1}^{\otimes}), \overline{\delta}[t/x] \vdash \sigma({s_2}^{\otimes})$, which gives us a proof of $\sigma({s_1}^{\otimes}), \forall x.\overline{\delta} \vdash \sigma({s_2}^{\otimes})$. This establishes the first clause of the definition of bisimulation.

Suppose that we have a proof of $\sigma({s_1}^{\otimes}), \forall x.\overline{\delta} \vdash \sigma(B^{\otimes})$; we must have a proof of $\sigma({s_1}^{\otimes}), \overline{\delta}[t/x] \vdash \sigma(B^{\otimes})$, and, by the assumption, we must have a situation s_2 with $B = {s_2}^{\otimes}$ and $\mathrm{Do}(\delta[t/x], s_1, s_2)$, which entails $\mathrm{Do}(\pi x.\delta(x), s_1, s_2)$. The third clause is similar.

Similarly for the proof of fourth caluse; we consider a proof similar to the above, but with multisets Γ and Δ in place of the $\sigma({s_1}^{\otimes})$ and $\sigma({s_2}^{\otimes})$, and apply the inductive hypothesis to show that both Γ and Δ must be non-empty. □

Remark 8. Levesque *et al.* are rather vague about the quantification employed here. If we wanted to have quantification bounded to some domain – say the domain of individuals introduced in Section 1.3.1 – we could define our translation as $\forall x.\mathcal{I}(x) \multimap \delta(x)$, and the above lemma would follow just as easily.

2.7 Nondeterministic Iteration

Levesque *et al.*'s definition is as follows:

$$\begin{aligned}\mathrm{Do}(\delta^*, s, s') \stackrel{\mathsf{df}}{=} \forall P.\Big[&\forall s_1.P(s_1, s_1) \wedge \\ &\forall s_1, s_2, s_3.\Big(P(s_1, s_2) \wedge \mathrm{Do}(\delta, s_2, s_3) \\ &\qquad \rightarrow P(s1, s3)\Big)\Big] \\ &\rightarrow P(s, s')\end{aligned} \tag{5}$$

which is a Peano-style induction.

We approach this as follows. We define $\overline{\delta^*}$ recursively, using Girard's fixed point theorem (see Section 0.2):

Definition 16.

$$\frac{\mathbf{Null} \,\&\, (\overline{\delta^*}; \overline{\delta}) \vdash}{\overline{\delta^*} \vdash}\,\mathsf{df}$$

Here **Null** is our null action.

We can now prove

Lemma 10. *Suppose that $\overline{\delta}$ bisimulates δ. Then $\overline{\delta^*}$ bisimulates δ^*.*

Proof. Suppose, for the first clause, that we have $\mathrm{Do}(\delta^*, s, s')$. We will apply Levesque *et al.*'s definition, with $\lambda s, s'.\, \sigma(s^{\otimes}),\ \overline{(\delta^*)} \vdash \sigma(s'^{\otimes})$ for their predicate $P(s, s')$; we prove the two conjuncts of the antecedent of their condition, and thus prove the consequent (i.e. $\sigma(s^{\otimes}),\ \overline{(\delta^*)} \vdash \sigma(s'^{\otimes})$).

$\forall \mathbf{s_1}.\mathbf{P(s_1, s_1)}$ This is $\sigma(s_1{}^{\otimes}),\ \overline{(\delta^*)} \vdash \sigma(s_1{}^{\otimes})$; we prove it as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\sigma(s_1{}^{\otimes}) \vdash \sigma(s_1{}^{\otimes})}}{\sigma(s_1{}^{\otimes}), \mathbf{Null} \vdash \sigma(s_1{}^{\otimes})}}{\sigma(s_1{}^{\otimes}),\quad \mathbf{Null} \& (\overline{(\delta^*)}; \overline{\delta}) \vdash \sigma(s_1{}^{\otimes})}}{\sigma(s_1{}^{\otimes}), \overline{(\delta^*)} \vdash \sigma(s_1{}^{\otimes})}\ \text{Definition 16}$$

Here the step labelled Definition 16 uses the left rule coming from the recursive definition.

$\forall \mathbf{s_1, s_2, s_3}.(\mathbf{P(s_1, s_2)} \wedge \mathbf{Do}(\delta, \mathbf{s_2, s_3}) \rightarrow \mathbf{P(s1, s3)}$ Suppose, then, that we have situations s_1, s_2 and s_3, and we have $P(s_1, s_2)$ and $\mathrm{Do}(\delta, s_2, s_3)$. With our choice of P, the first condition means that $\sigma(s_1{}^{\otimes}),\ \overline{(\delta^*)} \vdash \sigma(s_2{}^{\otimes})$; we can thus suppose that we have have a proof Π of this sequent. The second condition is that $\mathrm{Do}(\delta, s_2, s_3)$; by bisimulation for δ, we can suppose that we have a proof Π' of $\sigma(s_2{}^{\otimes}), \overline{\delta} \vdash \sigma(s_3{}^{\otimes})$. We have to prove $P(s1, s3)$, i.e. $\sigma(s_1{}^{\otimes}),\ \overline{(\delta^*)} \vdash \sigma(s_3{}^{\otimes})$, and we prove it as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Pi}{\vdots}}{\sigma(s_1{}^{\otimes}),\ \overline{(\delta^*)} \vdash \sigma(s_2{}^{\otimes})}}{\sigma(s_1{}^{\otimes}),\ \overline{(\delta^*)} \otimes \sigma(s_2{}^{\otimes})^{\perp} \vdash \perp} \qquad \dfrac{\dfrac{\Pi'}{\vdots}}{\sigma(s_2{}^{\otimes}),\ \overline{\delta} \vdash \sigma(s_3{}^{\otimes})}}{\overline{\delta} \otimes \sigma(s_2{}^{\otimes}) \vdash \sigma(s_3{}^{\otimes})}}{\sigma(s_1{}^{\otimes}),\ (\overline{(\delta^*)} \otimes \sigma(s_2{}^{\otimes})^{\perp}) \parr (\overline{\delta} \otimes \sigma(s_2{}^{\otimes})) \vdash \sigma(s_3{}^{\otimes})}}{\sigma(s_1{}^{\otimes}),\ \forall Z.(\overline{(\delta^*)} \otimes \sigma(Z)^{\perp}) \parr (\overline{\delta} \otimes \sigma(Z)) \vdash \sigma(s_3{}^{\otimes})}\ \text{Definition 13}}{\sigma(s_1{}^{\otimes}),\ \overline{\delta^*}; \overline{\delta} \vdash \sigma(s_3{}^{\otimes})}}{\sigma(s_1{}^{\otimes}),\ \mathbf{Null} \& (\overline{\delta^*}; \overline{\delta}) \vdash \sigma(s_3{}^{\otimes})}\ \text{Definition 16}}{\sigma(s_1{}^{\otimes}),\ \overline{\delta^*} \vdash \sigma(s_3{}^{\otimes})}$$

By the inductive definition of $\mathrm{Do}(\delta^*, s, s')$, we can now conclude $P(s, s')$; but, by our choice of P, this amounts to $\sigma(s^{\otimes}),\ \overline{(\delta^*)} \vdash \sigma(s'^{\otimes})$. So we have established one direction of the implication.

For the remaining clauses, we start with a proof of $\Gamma, \overline{\delta^*} \vdash \Delta$. We proceed by induction on the number of applications of Definition 16 in the proof.

We note, then, that, by the permutabilities, a proof of $\Gamma, \overline{\delta^*} \vdash \Delta$, can be assumed to start with the left rule for Definition 16, followed by one variant or the other of the left rule for &; so the proof must be equivalent to one of the

form

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\Pi\\ \vdots\\ \Gamma \vdash \Delta\end{array}}{\Gamma, \mathbf{Null} \vdash \Delta}}{\Gamma, \mathbf{Null} \,\&\, (\overline{\delta^*}; \overline{\delta}) \vdash \Delta}}{\Gamma, \overline{(\delta^*)} \vdash \Delta} \tag{6}$$

or of the form

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\Pi'\\ \vdots\\ \Gamma, \overline{\delta^*} \vdash \sigma(C), \Delta\end{array} \qquad \begin{array}{c}\Pi''\\ \vdots\\ \Gamma', \sigma(C), \overline{\delta} \vdash \Delta'\end{array}}{\Gamma, \Gamma', (\overline{\delta^*} \otimes \sigma(C)^{\perp}) \parr (\sigma(C) \otimes \overline{\delta}) \vdash \Delta, \Delta'}}{\Gamma, \Gamma', \forall Z.\, (\overline{\delta^*} \otimes \sigma(Z)^{\perp}) \parr (\sigma(Z) \otimes \overline{\delta}) \vdash \Delta, \Delta'}}{\Gamma, \Gamma', \overline{\delta^*}; \overline{\delta} \vdash \Delta, \Delta'}}{\Gamma, \Gamma', \mathbf{Null} \,\&\, (\overline{\delta^*}; \overline{\delta}) \vdash \Delta, \Delta'}}{\Gamma, \Gamma', \overline{(\delta^*)} \vdash \Delta, \Delta'} \tag{7}$$

For the second clause of the definition of bisimulation, suppose that we have, on the left, $\lbag \sigma(s^{\otimes}) \rbag$ and, on the right, $\lbag \sigma(B) \rbag$. If we have a proof of the first sort, then we must have $B = s^{\otimes}$, up to permutation and units; this establishes the clause. If we have a proof of the second sort, then we argue as follows. Π' has fewer applications of the left rule for $\overline{\delta^*}$, so inductively we can assume bisimulation for that. We have $\Gamma \sqcup \Gamma' = \lbag \sigma(s^{\otimes}) \rbag$; however, because of bisimulation for Π', we must have Γ nonempty, so $\Gamma = \lbag \sigma(s^{\otimes}) \rbag$ and $\Gamma' = \emptyset$. Similarly, $\Delta \sqcup \Delta' = \lbag \sigma(B) \rbag$ and, because of bisimulation for Π'', we must have Δ' nonempty. So $\Delta = \emptyset$ and $\Delta' = \lbag \sigma(B) \rbag$. This means that, because of bisimulation for Π', there is a situation s'' with $C = s''^{\otimes}$ and $\mathrm{Do}(\delta^*, s, s'')$. Now because of bisimulation for Π'', there is a situation s' with $B = s'^{\otimes}$ and $\mathrm{Do}(\delta, s'', s')$. Finally, because of the GOLOG semantics of δ^*, we must have $\mathrm{Do}(\delta^*, s, s')$; this establishes the second clause. The third clause is very similar.

For the fourth clause, we proceed by cases. If we have a proof of the first sort, then we have the result by properties of the null action; if we have a proof of the second sort, then, since Π' has fewer applications of the left rule for $\overline{\delta^*}$, we can assume the result inductively. So by bisimulation for Π', we must have Γ nonempty. by bisimulation for Π'', we must have Δ' nonempty. □

3 Procedures

Levesque *et al.* have a contextually defined semantics for the definition and use of procedures. Suppose we have a procedure with head $P(\overrightarrow{v})$ and body δ; we write this proc $P(\overrightarrow{v})$ δ endProc. Suppose that we have definitions for procedures $P_1, \ldots, P_n$, and that we also have a program $\delta_0(X_1, \ldots, X_n)$ (where the X_i

are suitable variables), then Levesque *et al.* give the following semantics:

$$\begin{aligned}
&\mathrm{Do}\left(\left\{\begin{array}{l}\mathsf{proc}\ P_1(\overrightarrow{v}_1)\ \delta_1[P_1,\ldots,P_n,\overrightarrow{v}_1]\ \mathsf{endProc};\cdots;\\ \quad \mathsf{proc}\ P_n(\overrightarrow{v}_n)\ \delta_n[P_1,\ldots,P_n,\overrightarrow{v}_n]\ \mathsf{endProc};\\ \quad \delta_0[P_1,\ldots,P_n]\end{array}\right\},s,s'\right)\\
&\stackrel{\mathsf{df}}{=}\ \forall Y_1,\ldots,Y_n.\left[\bigwedge_{i=1}^{n}\left(\forall s_1,s_2,\overrightarrow{v_i}.\mathrm{Do}(\delta_i[Y_1,\ldots,Y_n,\overrightarrow{v_i}],s_1,s_2)\rightarrow \mathrm{Do}(Y_i(\overrightarrow{v_i}),s_1,s_2)\right)\right]\\
&\quad\rightarrow \mathrm{Do}(\delta_0[Y_1,\ldots,Y_n],s,s') \qquad (8)
\end{aligned}$$

This needs a little supplementary explanation.

Firstly, we need the notion of applying $\mathrm{Do}(\cdot,\cdot,\cdot)$ to a predicate. Here the Y_i are predicates of suitable arity – the arity of Y_i must be the arity of P_i plus two – and we must, in addition, define $\mathrm{Do}(Y(\overrightarrow{v}),s,s')$, for such a predicate Y:

$$\mathrm{Do}(Y(\overrightarrow{v}),s,s') \quad\stackrel{\mathsf{df}}{=}\quad Y(\overrightarrow{v},s,s'). \qquad (9)$$

Secondly, the procedure definitions are, in general recursive, so the procedure bodies (δ_i) should have argument places for the P_i, and, of course, for the parameters $\overrightarrow{v}_i$ passed to the procedure. The calling procedure (δ_0), of course, has argument places only for the P_i.

3.1 Contexts and Substitutions

The above definition uses the idea of substituting a procedure into a context. We need a technical lemma; for simplicity, we will prove these for the definition of only one procedure, but the simultaneous recursive definition of several procedures can be handled in an entirely similar way.

The lemma, then, shows how to relate substitution and bisimulation. First a definition.

Definition 17. If Θ is a linear logic formula, then the predicate Y_Θ is defined by

$$Y_\Theta(s,s') \quad\stackrel{\mathsf{df}}{=}\quad \sigma(s^\otimes),\Theta\ \vdash\ \sigma(s'^\otimes).$$

We extend the linear logic translation of GOLOG formulae to formulae involving free propositional variables: we simply translate the variable Y by itself (or by a suitably chosen free propositional variable in linear logic, if one insists on disjoint sets of variables). We now have

Lemma 11. *Let $\delta[Y]$ is a* GOLOG *action with a free propositional variable, and let Θ be a (ground) linear logic formula such that any valid proof of $\Gamma,\Theta\vdash\Delta$ – where Γ and Δ are multisets of propositions of the form $\sigma(\cdot)$ – must have Γ and Δ non-empty. Then $\delta[Y_\Theta]$ bisimulates $\overline{\delta}[\Theta]$.*

Proof. We can prove this by an induction on the logical complexity of δ. Since this is a GOLOG action, it must be built from basic actions and the variable Y using the action combinators. So there are the following cases:

Basic Actions We use Proposition 2.

Free Variables In this case we have to prove that $\mathrm{Do}(Y_\Theta, s, s')$ bisimulates $\sigma(s^\otimes)$, $\Theta \vdash \sigma(s'^\otimes)$ and satisfies Property A, but this is immediate from the definition of Y_Θ and (for the fourth clause of the definition of bisimulation) the the assumption on Θ.

Test Actions We use Lemma 6.

Sequence We use Lemma 7.

Choice of Actions We use Lemma 8.

Choice of Arguments We use Lemma 9.

Iteration We use Lemma 10.

□

3.2 Translating Procedure Definitions

We make the following definition: again, we restrict ourselves, for the sake of simplicity, to the definition of a single procedure.

Definition 18. Given a procedure definition proc $P(\overrightarrow{v})$ $\delta(\overrightarrow{v})[P]$ endProc, we define the linear translation, $\overline{P}$, by the left clause $\dfrac{\overline{\delta}(\overrightarrow{v})[P] \vdash}{\overline{P}(\overrightarrow{v}) \vdash}$ df.

Then, given a program together with a series of procedure definitions, we simply add the translations of the procedure definitions to our stock of definitions, and translate the procedure in the normal way; calls of procedures defined in the environment can now be translated, since we have clauses for them.

Lemma 12. *If $\overline{P}[\overrightarrow{v}]$ is an instatiation of the translation of a* GOLOG *procedure, and if $\Gamma, \overline{P}[\overrightarrow{v}] \vdash \Delta$ is a valid sequent, where Γ and Δ are multisets of formulae of the form $\sigma(\cdot)$, then Γ and Δ are non-empty.*

Proof. We prove this by induction on the number of applications of the left rule for $\overline{P}$. By the permutabilities, we may assume that the proof starts as follows:

$$\dfrac{\begin{array}{c}\Pi \\ \vdots \\ \Gamma, \overline{\delta(\overrightarrow{v})}[P] \vdash \Delta\end{array}}{\Gamma, \overline{P(\overrightarrow{v})} \vdash \Delta}$$

We can argue inductively and reduce the problem to that of proving the result for proofs of smaller size. Using Lemmas 7, 8, 9 and 10 we can reduce the problem to proving the result for basic actions, test actions, and applications of the left rule for P. The first two are clear, by previously established results: and the applications of the left rule for P belong to subproofs with fewer applications of that rule, so we can assume the result for them inductively. □

3.3 Bisimulation with Procedure Definitions

We now prove our bisimulation result – again, for only one recursively defined predicate; however, it can be easily extended to the case where we have several such.

First a technical lemma:

Lemma 13. *Suppose that we have a procedure definition*

$$\mathit{proc}\ P_(\overrightarrow{v})\ \ \delta(\overrightarrow{v})[P]\ \ \mathit{endProc}.$$

Then, for all $\overrightarrow{v}$*, and all* s, s'*, we have*

$$Do(\delta(\overrightarrow{v})[Y_{\overline{P}}], s, s') \rightarrow Do(Y_{\overline{P}}, s, s')$$

Proof. Suppose that we have $\mathrm{Do}(\delta(\overrightarrow{v})[Y_{\overline{P}}], s, s')$. By Lemma 11, we have $\sigma(s^{\otimes}), \overline{\delta(\overrightarrow{v})[P]} \vdash \sigma(s'^{\otimes})$. But this gives us a proof of $\sigma(s^{\otimes}), \overline{P} \vdash \sigma(s'^{\otimes})$, viz:

$$\dfrac{\begin{array}{c}\Pi \\ \vdots \\ \sigma(s^{\otimes}), \overline{\delta(\overrightarrow{v})[P]} \vdash \sigma(s'^{\otimes})\end{array}}{\sigma(s^{\otimes}), \overline{P} \vdash \sigma(s'^{\otimes})}\ \dagger$$

using the left rule for P at $\dagger$. By definition of $Y_{\overline{P}}$, we thus have $\mathrm{Do}(Y_{\overline{P}}, s, s')$. □

Lemma 14. *Suppose that we have a procedure* proc $P_(\overrightarrow{v})\ \delta_1(\overrightarrow{v})[P]$ *endProc, and suppose also that we have a* GOLOG *action* $\delta_0[Y]$ *with a free variable. Then* $\overline{\delta_0[P]}$ *bisimulates* proc $P_(\overrightarrow{v})\ \delta_1(\overrightarrow{v})[P]$ *endProc*; $\delta_0[P]$.

Proof. Suppose, for the first clause, that

$$\mathrm{Do}(\mathsf{proc}\ P_(\overrightarrow{v})\ \delta_1(\overrightarrow{v})[P]\ \mathsf{endProc}; \delta_0[P], s, s').$$

By definition, we have

$$\begin{aligned}&\mathrm{Do}\left(\left\{\mathsf{proc}\ P(\overrightarrow{v})\ \delta_1[P]\ \mathsf{endProc}\ ;\ \delta_0[P]\right\}, s, s'\right)\\ &\quad\stackrel{\mathrm{df}}{=}\ \forall Y.\left(\forall s_1, s_2, \overrightarrow{v}.\mathrm{Do}(\delta_i[Y], s_1, s_2) \rightarrow \mathrm{Do}(Y(\overrightarrow{v}), s_1, s_2)\right)\\ &\qquad\rightarrow \mathrm{Do}(\delta_0[Y], s, s')\end{aligned} \tag{10}$$

Now substitute, for Y, the $Y_{\overline{P}}$ of Definition 17. By Lemma 13, the antecedent of the *definiens* is true; thus we have $\mathrm{Do}(\delta_0[Y_{\overline{P}}], s, s')$. By Lemma 11 (this time applied to δ_0), we have $\sigma(s^{\otimes}), \overline{\delta_0[P]} \vdash \sigma(s'^{\otimes})$.

For the remaining clauses, suppose that we have $\Gamma, \overline{\delta_0[P]} \vdash \Delta$. Suppose now that $\Gamma = \wr\sigma(s^{\otimes})\wr$, for some situation s, and that $\Delta = \wr\sigma(B)\wr$ is a singleton of the appropriate sort. Inductively we can assume bisimulation for Π, since it contains fewer applications of the left rule for P. So we know that $B = s'^{\otimes}$ for some situation s'; this is more bureacratic than one might suppose, since the GOLOG semantics for actions involving procedure definitions is not *prima facie*

compositional in the same way as the other definitions are. So we suppose that we have a predicate Y such that

$$\forall s_1, s_2, \overrightarrow{v}. \mathrm{Do}(\delta_1(\overrightarrow{v})[Y], s_1, s_2) \rightarrow \mathrm{Do}(Y(\overrightarrow{v}), s_1, s_2) \tag{11}$$

We must show that $\mathrm{Do}(\delta_0[Y], s, s')$.

We argue by induction on the complexity of δ_0: that is, we prove that, if we have a proof of $\sigma(\delta_0[P]), s^{\otimes} \vdash \sigma(s'^{\otimes})$, and a predicate Y satisfying (11), that we have $\mathrm{Do}(\delta_0[Y], s, s')$.

If δ_0 is a primitive action or a test action, then it does not involve P and we have already handled this case. If $\delta_0[P] = P(\overrightarrow{v})$, for some $\overrightarrow{v}$, we can assume, by the permutabilities, that the proof starts

$$\dfrac{\begin{array}{c}\Pi \\ \vdots \\ \Gamma, \overline{\delta_1(\overrightarrow{v})[P]} \vdash \Delta\end{array}}{\Gamma, \overline{\delta_0[P]} \vdash \Delta}$$

where Π is a proof with fewer applications of the left rule for P than the original proof; so inductively we can assume the result for $\sigma(s^{\otimes}), \overline{\delta_1(\overrightarrow{v})} \vdash \sigma(s'^{\otimes})$. We thus have $\mathrm{Do}(\delta_1(\overrightarrow{v})[Y], s, s')$. By the assumption on Y, we have $\mathrm{Do}(Y(\overrightarrow{v}), s, s')$, which is what we wanted to establish.

Finally, there are the inductive cases. Consider, for example, a sequence of actions. If $\delta_0[Y] = \delta_0'[Y]; \delta_0''[Y]$, then, by Lemma 7, any proof of

$$\sigma(s^{\otimes}), \overline{\delta_0'[P]; \delta_0''[P]} \vdash \sigma(s'^{\otimes})$$

must come from proofs of $\sigma(s^{\otimes}), \overline{\delta_0'[P]} \vdash \sigma(s''^{\otimes})$ and of $\sigma(s''^{\otimes}), \overline{\delta_0''[P]} \vdash \sigma(s'^{\otimes})$. Assuming the result for δ_0' and δ_0'', we have $\mathrm{Do}(\delta_0'[Y], s, s'')$ and $\mathrm{Do}(\delta_0''[Y], s'', s')$; by the semantics of the ; operator, we have $\mathrm{Do}(\delta_0'[Y]; \delta_0''[Y], s, s')$, which is what we have to establish.

The other inductive cases are entirely similar.

That establishes the second clause of the definition of bisimulation; the third clause is entirely similar. The fourth clause is much simpler, because it does not involve establishing anything of the form $\mathrm{Do}(\delta_0[Y], s, s')$. □

3.4 Locally Defined Procedures

Levesque *et al.* mention that

> [b]y using programs as above within the bodies of other procedures, we obtain the tree-structured nesting of procedures typical of Algol-like languages. Moreover, we get the lexical scoping rules of these languages for free from our use of quantifiers in the definition of D_0. [6, p. 8n]

We have to show, then, that we can give local scope to procedure definitions.

We do this as follows: the trick is quite general, and can be used for restricting the scope of all manner of entities. Suppose we have, as above, a procedure definition

$$\mathsf{proc}\ P(\overrightarrow{v})\ \delta_1(\overrightarrow{v})[P]\ \mathsf{endProc};$$

and we have, correspondingly, its linear logic translation, $\overline{P}$. We can now define a higher-order predicate $\hat{P}(\cdot)$ as follows:

$$\frac{\mathbf{1} \vdash}{\hat{P}(\overline{P}) \vdash} \mathsf{df} \tag{12}$$

We now have

Lemma 15. *If* proc P δ_0 endProc *is a* GOLOG *procedure, and* $\delta_0[X]$ *is a* GOLOG *context, then*

$$\sigma(s^{\otimes}),\ \overline{\delta_0}[\overline{P}] \ \vdash\ \sigma(s'^{\otimes}) \tag{13}$$

iff

$$\sigma(s^{\otimes}),\ \hat{P}(X) \otimes \overline{\delta_0}[X] \ \vdash\ \sigma(s'^{\otimes}) \tag{14}$$

Proof. Notice first that, by the permutativities, a proof of (14) can be assumed to be of the form

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\Pi \\ \vdots \\ \sigma(s^{\otimes}),\ \overline{\delta_0}[X] \ \vdash\ \sigma(s'^{\otimes})\end{array}}{\sigma(s^{\otimes}),\ \mathbf{1}, \overline{\delta_0}[X] \ \vdash\ \sigma(s'^{\otimes})}}{\sigma(s^{\otimes}),\ \hat{P}(\overline{P}), \overline{\delta_0}[X] \ \vdash\ \sigma(s'^{\otimes})}}{\sigma(s^{\otimes}),\ \hat{P}(\overline{P}) \otimes \overline{\delta_0}[X] \ \vdash\ \sigma(s'^{\otimes})}$$

so the result is clear. □

We now restrict scopes as follows. Suppose that we have a block of any sort (which need not necessarily be inside a procedure body); we can write it

$$\{\mathsf{proc}\ P\ \delta_1[P]\ \mathsf{endProc} \ldots \delta_0[P] \ldots\}\,;$$

its linear logic translation will then be

$$\exists X.\left(\ldots \hat{P}(X) \otimes \overline{\delta_0}[X] \ldots\right)$$

and some routine work with environments will show that this, indeed, gives the correct scope to the procedure definition.

4 The Result

Putting all this together, we now have:

Theorem 1. *If* δ_0 *is a* GOLOG *program, then* $\overline{\delta_0}$ *bisimulates* δ.

5 Assessment

We have shown that the linear logic formulation of GOLOG is, to all intents and purposes, equivalent to the original definition. This means that, if we have a well-motivated extension of the linear logic formulation, we can happily use that, regardless of whether the original definition can cope with it.

5.1 Second Order Entities

Levesque *et al.* have a rather equivocal attitude to higher order entities. Firstly, they seem to confuse the computer science notion of a "first class entity" with that of being first-order:

> why do we not treat [complex actions] as first class objects (terms) in the language of the situation calculus? ... To see what must happen if we avoid the macro approach, suppose we treat complex actions as genuine first order terms in the language of the situation calculus. [6, Section 3.2]
>
> By opting to define programs as macros, we obtain a much simpler theory than if we were to reify those actions. The price we pay for this is a much less expressive formalism. For example, we cannot quantify over complex actions ... [6, Section 3.3]

So the direction of thought seems clear: the only first class objects are those we can quantify over, and those are first-order objects.

This seems very strange. For one thing, Levesque *et al.* themselves resort to higher order quantification, namely in the definition of the semantics of procedures – (8) – where the Y_i are higher order entities and are clearly quantified over. Furthermore, the idea of a "first class entity" is borrowed from mainstream computer science, and it is there used without any prejudice against the higher order: for example, Stoy [14, p. 39] describes first class entities as those which can be assigned to variables, passed as parameters, returned as results of function calls, and so on. None of this is prejudiced against the higher order, and, of course, functional programming languages explicitly treat functions – which are higher order if anything is – as first class entities. And this lack of prejudice is useful: Abelson and Sussman, for example [1, Chapter 1], start their textbook with an extensive and well-motivated series of examples in which higher order entities are used and passed from one procedure to another.

Correspondingly, our linear logic formalism is not prejudiced against higher-order entities: the logic was originally formalised as a second order theory [3]. We have ourselves used some higher-order language in our treatment of local procedure definitions, and there will probably be a useful role for more explicit higher-order operators. For example, Davidson [2] introduces existential quantification over actions and events on the grounds that this is a natural account of the structure of ordinary language.

5.2 The Algorithm

Levesque *et al.* have an algorithm which is fairly explicitly forward-directed. Our formulation, in terms of proof search, is not biased in that way; we can use the methods of Dale Miller [9] to give it a proof search algorithm. An algorithm which could be used bidirectionally would be much more useful: we could use it, for example, for explanation and planning as well as for prediction.

5.3 Other Fixed Points

Girard's Fixpoint Theorem is not limited to providing merely the least fixpoint, and there are occasions when we might be interested in others. For example, if

we are interested in something like Kowalski's cycle predicate, we would want to construct a greatest fixpoint of some appropriate operator. We could also define such a thing by a suitable linear logic definition; the proper treatment of this would involve dealing with infinite proof objects, as in [8]. The linear logic treatment, then, could deal with this: Levesque *et al.*'s, with its bias towards least fixpoints, would probably face severe difficulties.

References

[1] Harold Abelson, Gerald Jay Sussman, and Julie Sussman. *Structure and Interpretation of Computer Programs.* MIT Press, Cambridge, MA, second edition, 1996.

[2] Donald Davidson. The logical form of action sentences. In *Essays on Actions and Events*, pages 105–148. Oxford University Press, 1980. Originally published in N. Rescher (ed.), *The Logic of Decision and Action*, University of Pittsburgh Press, 1967.

[3] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[4] Jean-Yves Girard. A fixpoint theorem in linear logic. A message posted on the `linear@cs.stanford.edu` mailing list, available at `http://www.csl.sri.com/linear/mailing-list-traffic/www/07/mail_3.html`, February 1992.

[5] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.

[6] Hector J. Levesque, Raymond Reiter, Yves Lespérance, Fangzehn Lin, and Richard B. Scherl. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming*, 19, 20:1–25, 1994.

[7] Fangzhen Lin and Yoav Shoham. Provably correct theories of action (preliminary report). In *AAAI–91: Proceedings, Ninth National Conference on Artificial Intelligence*, volume 1, pages 349–354, Menlo Park etc., 1991. American Association for Artificial Intelligence, AAAI Press/MIT Press.

[8] Raymond McDowell, Dale Miller, and Catuscia Palamidessi. Encoding transition systems in sequent calculus: Preliminary report. *Electronic Notes in Theoretical Computer Science*, 3, 1996. Available from `http://www.elsevier.nl/locate/entcs/volume3.html`.

[9] Dale Miller. Forum: A multiple-conclusion specification language. *Theoretical Computer Science*, 165:201–232, 1996.

[10] Richard Montague. *Formal Philosophy: Selected Papers of Richard Montague.* Yale University Press, New Haven, CT, 1974. R. Thomason (ed.).

[11] Richard Montague. The proper treatment of quantification in ordinary English. In *Formal Philosophy: Selected Papers of Richard Montague* [10], pages 247–270. R. Thomason (ed.).

[12] John Reynolds. Syntactic control of interference. In *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages*, pages 39–46. Association for Computing Machinery, 1978.

[13] Peter Schroeder-Heister. Cut-elimination in logics with definitional reflection. In D. Pearce and H. Wansing, editors, *Nonclassical Logics and Information Processing*, volume 619 of *Lecture Notes in Artificial Intelligence*, pages 146–171, Berlin, etc., 1992. Springer-Verlag. International Workshop, Berlin 1990.

[14] Joseph E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory.* MIT Press, Cambridge, MA, 1977.

[15] Richmond H. Thomason. Introduction. In *Formal Philosophy: Selected Papers of Richard Montague* [10], pages 1–69. R. Thomason (ed.).

[16] Graham White. A linear meta-interpreter for the situation calculus. Submitted to the *Journal of Logic and Computation.*

[17] Graham White. The design of a situation-based Lygon metainterpreter: I. simple changes and persistence. Technical Report 729, Department of Computer Science, Queen Mary and Westfield College, London, October 1996.