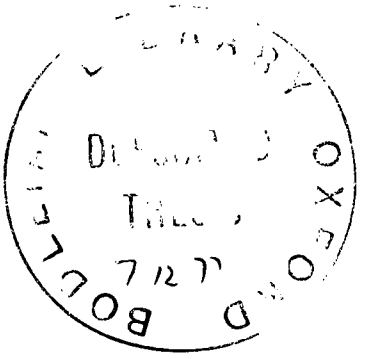

A Rigorous Examination Of Sequentially Expressed Algorithms To Reveal Their Concurrency Hierarchies

C.M.Jones

(St. Peter's College, Oxford)

A thesis submitted for the degree of Doctor of Philosophy

University of Oxford *July 1977*

Abstract

The methods of Scott-Strachey semantics are applied to the problem of writing programs for parallel computers, using serial languages such as are in common use, with compilers which attempt to discover and exploit potential parallelism in independent sections of program.

A mathematical model of a parallel computer is first developed in detail, and in chapters 1 and 2 three basic conditions are derived which together ensure determinacy of operation in a parallel machine, first in purely semantic terms, then in a form more related to the syntactic structure of a language.

The remaining chapters apply these basic conditions to three actual languages, showing how the conditions can be reduced to purely syntactic (hence compiler evaluable) tests. Chapter 3 uses a very simple language to introduce the techniques required. Chapter 4 adds procedures, with all the complexity they involve. Chapter 5 considers the rather more fruitful area of arrays and loops.

The result, in all but the most trivial cases, is a rather complicated set of conditions which, while they produce the desired effect, suggest that today's commonly used languages are not the most suitable method of exploiting parallel computers.

Contents

Chapter 0: Introduction

"'What is the use of a book,' thought Alice, 'without pictures or conversations?'"

Alice's Adventures in Wonderland: Chap 1.

The rapidly increasing quantity of work that is being found for computers to do, together with the realisation that there are physical limits on the speed at which a computer can be made to operate which are likely soon to make themselves felt, has led to the development of, and considerable growth of interest in, multi-processing computers; i.e. machines with several processing units (of similar or different kinds) which can act simultaneously, often on a common set of data. The earliest use of such machines gave each processor a separate program to execute, which it did completely independently of the other processors. This is not a very interesting technique, since it is virtually equivalent to using a number of small, completely separate computers, and it also has the disadvantage of requiring large quantities of storage for many programs to be available at one time. More recently, the problem of applying a large number of interacting processors to different parts of a single program has been considered. Needless to say, this poses considerable problems, not least for the programmer. Geller and Weinberg [12] suggest the principle: "Parallel is more natural than serial, unless proved otherwise", with the corollary: "Unordered is more natural than ordered, unless proved otherwise"; unfortunately, this very easily becomes: "Chaos is more natural than order...", which may sometimes be true, but it is hardly desirable! The problem with a program

with interacting parallel processes is that the result may well depend on the relative timings of the interactions, so that two runs of the same program with the same data may produce different results. This problem rarely occurs with serial programs, since, although some programming languages do not define the order of certain operations (e.g. evaluation of subexpressions), most implementations of these languages in practice do. Furthermore, the problem is worse than that of simply reordering commands or expressions of a program; for instance, the two statements:

```
x := x + 1 ;    x := x + 2
```

can be executed serially in either order and will always give the same result; if, however, they are executed in parallel, the outcome may be to increment x by 1, 2 or 3!

It seems only reasonable then, that a programmer should be able to be sure that any program he writes is at least determinate - his job is difficult enough even with such assurance. One can distinguish three approaches to programming parallel computers:

i) The use of languages with explicit manipulation of parallel processes, but where the constructions used are guaranteed to produce determinate results, e.g. Data Flow languages [2] [11] [16]. Unfortunately, it is not at all clear how best to do this, and the work in this area is in early stages - most of it is in the state where "programs" consist of flowcharts which then have to be hand-coded into a machine code. Also, this approach has a tendency (at least in its current state of development) to produce languages very much oriented to particular hardware (most of which does not exist). So, while

this approach may have considerable promise for the future, it will not be considered further here.

ii) The use of parallel languages (or serial languages extended to contain parallel features) whose constructs are not such as to guarantee determinate behaviour. The advocates of this approach (see [3] [12] [25] [37]) do not seem to be concerned about guaranteeing determinacy. But since the language does not guarantee it, and by the philosophy outlined above it is unreasonable to expect the programmer to do so, the only alternative is for the compiler to check that any instance of a parallel constuct is in fact deterministic. This makes the definition of what is a valid program unduly complicated, and the method has little to commend it over the third approach.

iii) The use of conventional serial languages with a compiler which is capable of deciding that certain sections of a program, although not specified as such by the programmer, are in fact independent, and can be executed in parallel. This inevitably makes the compiler rather large and slow, but the approach has (at least at present) the two-fold attraction that not only are a great many existing programs written in such languages, but also serial programming is (comparatively) well understood.

The basis of the difference between the first two approaches and the last one is the question of which is more effective at finding and exploiting potential parallelism in a problem, computer or human programmer. The ideal answer is almost certainly a carefully balanced combination; the third method is unlikely to discover all the potential parallelism of an algorithm, and hasn't a hope of finding a better algorithm for the same problem; on the other hand, there is much that a

machine can do, and do it more thoroughly and reliably. Having said that, the remainder of this work will be devoted to the consideration of approach (iii) - the automatic discovery, i.e. by a compiler, of parts of a sequentially expressed algorithm which can be executed concurrently while retaining the determinacy of the algorithm.

Much work has already been done in this area; a survey of some of it, together with some other aspects of parallel processing, may be found in [4]. Many people have considered arithmetic expressions to see how they can be rearranged, using the commutativity, associativity and even distributivity of the operators, to make the best use of parallelism [6] [7] [9] [14] [17] [18] [24] [28] [31] [36]. Indeed, quite an inordinate amount of effort seems to have been expended in this particular area, bearing in mind that most programs only contain very short, simple expressions. The other main area of work has been in discovering whole statements or blocks of statements which are independent and can therefore be exceuted concurrently. [5] [8] [14] [29] have considered this aspect; [24] takes several special cases where this technique can usefully be employed, the most interesting and profitable of which is the case of loops where different iterations of the loop can be executed in parallel; see also [20] [30]. Kuck et al [19] have found some quite encouraging empirical results concerning the usefulness of these techniques in terms of the speed-up of programs.

The approach used herein will be somewhat different; most of the work described above was pragmatic (except for some of the more esoteric work on expressions) in that it was directed

to extracting the maximum amount of parallelism from existing programs (usually written in Fortran). The present work is more rigorous; starting from a formal description of a language's semantics and a semantic model of a parallel computer, methods are developed to find which sections of a program can be evaluated concurrently. Moreover, the analysis will follow the recursive structure of the program, so that a hierarchy of possible parallelism may be detected, rather than the single level that is generally only possible when analysing Fortran programs. Transformations of the program, as in [20] [24], will not be considered, nor will questions of scheduling arbitrary numbers of tasks on a fixed number of processors (see [1]), or of how large and numerous tasks should be to make optimum use of any given machine. One of the implications of this approach is that there is little point in considering a language like Fortran, or even Algol - their semantic descriptions are clumsy, and many interesting features are either absent or obscured behind irrelevant quirks of the language. Instead, three languages have been designed to exemplify the methods. Between them they (hopefully) include most of the significant features of existing languages, including ones like the sharing of one memory location by two different identifiers - an important problem which has been almost totally forgotten or ignored - [8] and [14] mention it in passing, but then ignore it.

The first, described in chapter 3, has only very basic features, and its potential for parallelism is small - the opportunities appearing in, for example, compound expressions (which are rarely complex enough to allow much parallel activity) and the sequencing of commands in a block (which are

rarely completely independent). It does, however, serve to introduce the steps and techniques required to prove the desired results.

Chapter 4, with the second language, considers procedures and some of their associated problems. Unfortunately, the opportunities for parallelism are not significantly more, but this chapter attempts to consider realistically some of the complexity involved in applying the methods to familiar types of language.

The third language (see chapter 5) branches out in a different direction, and by considering loops in more detail, an area of considerably greater promise for useful parallelism is explored.

The methods for describing the semantics are essentially those of Scott-Strachey semantics as expounded in [23]; details of notation, etc., will be discussed as required.

For the purpose of modelling it, a parallel-processing computer can be envisaged as a number of independent processing units, (all similar, or each with a different special function,) probably some common storage, (this can be thought of as one of the processors, but it is convenient not to,) and some means of communication between pairs of processors and between processors and the store. The action of each processor (though not necessarily the time it takes for that action) is a known function purely of the communications it has sent and received in the past. Thus the action of the machine is entirely determined by the state of the store and communication lines.

The (not unreasonable) assumption must be made that the hardware of the machine prevents from happening simultaneously

two or more communications which clash (e.g. attempts by two different processes to access the same memory location); and clearly two communications which have no connection with each other may be considered one at a time, even though they occur simultaneously.

Thus it is reasonable to consider communications one at a time, as if nothing else were happening; and, with each communication, any computation done by the processor (since its last communication) as if squashed into the time since the last communication of the whole machine.

Thus each processor may be considered as executing a sequence of transformations of the state of the machine, which is interleaved with similar sequences from other processors. Of course, later transformations may well depend on the state of the machine which is being affected by all the processors, and therefore on the transformations executed by other processors and the way in which they are interleaved.

So much for a general description of the model. The next chapter will develop the model in detail and prove some results about conditions in terms of abstract semantics; chapter 2 will give further general results; and chapters 3, 4 and 5 will describe the three exemplary languages and develop syntactic (i.e. compiler-usable) conditions for the use of parallelism.

CHAPTER 1: SEMANTIC CONDITIONS

"'If everybody minded their own business,' the Duchess said, in a hoarse growl, 'the world would go round a deal faster than it does.'"

Alice's Adventures in Wonderland: Chap 6.

1.0 Introduction and Notation

This chapter will be concerned with defining the details of the semantic model, and with deriving three basic conditions which ensure the determinacy of parallel computation. Later chapters will replace these semantic conditions by syntactic ones which can be applied directly (by a compiler) to a program text.

The mathematical structures used will be complete lattices; for a detailed discussion of these, their use in programming language semantics, and their construction from recursive equations, see [23]. The notation used herein will be similar to that of [23]. Lambda notation will be widely used in defining functions. Semantic domains will be denoted by single upper-case Roman letters, and their elements by corresponding lower-case Greek letters, using primes and subcripts to distinguish different elements of the same domain, thereby usually eliminating the need to state explicitly to which domain an object belongs; thus $\varepsilon, \varepsilon', \varepsilon_0, \varepsilon_i'$ will all be understood to be elements of E, ϕ of F, etc., but note the three cases $\theta \in C$, $\eta \in H$, and, because of the special function (so to speak) of λ in defining functions, $\alpha \in L$. When elements of anonymous domains are required, their functionalities will normally be given

explicitly. The one exception to the unrestricted use of primes and subscripts will be in the case of the domain K: since three similar domains K, K′ and K″ are required, κ,κ_0, etc. will be used for elements of K, κ',κ_0', etc. for K′, and κ'',κ_0'' etc. for K″. Italicised names will be used for specific functions, and (occasionally) specific atomic objects (eg *True*, *False*). With reference to domains, +, ×, → (indicating continuous functions - see [23]), ϵ have their usual meanings, [] are used for bracketting, and {} are used to enclose a specific enumeration of the elements of a domain. Precisely what definitions of + and × are adopted is largely irrelevant, though care needs to be taken occasionally to ensure that all the functions used are continuous.

Finite lists of elements of a domain D will be denoted by $\delta^*\epsilon D^*$; such lists can be specified either, as in the case of products, by a sequence enclosed in the special brackets ⟨⟩,

eg. $\langle \delta_1,\delta_2,\ldots,\delta_\nu\rangle$

or by use of the symbol ⅄ which is used similarly to the summation symbol $\sum$ of conventional mathematics,

eg. $\bigtimes_{\iota=1}^{\nu}\delta_\iota=\langle \delta_1,\delta_2,\ldots,\delta_\nu\rangle$

$\#\delta^*$ denotes the number of components of δ^*; selection of the ιth element is denoted by $\delta^*\downarrow\iota$, often abbreviated (when this is not ambiguous) to δ_ι; $\delta^*\uparrow\nu$ denotes the list produced by removing the first ν elements of δ^*; $\delta_1^*\hat{}\delta_2^*$ denotes the concatenation of δ_1^* and δ_2^*, and ⋀ is used for multiple concatenation in the same way as ⅄ and $\sum$; $\delta\triangleleft\delta^*$ indicates that δ is a component of δ^*; $\delta_1^*\backslash\delta_2^*$ denotes the list formed from δ_1^* by omitting all the components of δ_2^* wherever they occur; note that if $P\delta$ is a continuous predicate, then $(\forall\delta\triangleleft\delta^*,P\delta)$ and

$(\exists\delta{\lhd}\delta^*, P\delta)$ are also continuous. The expression $\tau{\rightarrow}\varepsilon_1,\varepsilon_2$ where τ is an element of the truth value domain T, takes the value ε_1 if $\tau=\mathit{True}$, ε_2 if $\tau=\mathit{False}$, $\perp$ if $\tau=\perp$, and $\top$ if $\tau=\top$. If $\delta\epsilon[A{\rightarrow}B]$, then $\delta[\beta/\alpha]$ denotes $(\lambda\alpha'.\alpha=\alpha'{\rightarrow}\beta,\delta\alpha')$. In describing many functions, a recursive equation will be given, in the hope that this will be clearer than, but easily reformulated as, a precise definition in terms of the least fixed point operator *Fix*, to which the given equation will be considered equivalent. In the statement of lemmas and theorems, "free" variables should of course be assumed to be implicitly universally quantified over the appropriate domains; and within proofs, the symbols $\mathcal{L}$ and $\mathcal{R}$ will denote the left- and right-hand sides respectively of the result to be proved. Parentheses () and {} will be used (hopefully) to make the parsing of expressions unambiguous and as clear as possible: the author apologises for any lapses in this respect.

1.1 Basic Domain Structures

This section will describe the basic semantic domains and manipulating functions which form the foundation of the model, and will prove a number of basic reults about these functions.

First, $\nu\epsilon N$ and $\iota\epsilon I$ will normally denote non-negative and positive integers respectively.

A domain S is needed to model the "state" of the machine - this corresponds to the store of serial semantics - it will have components representing the memory of a serial machine, but also others unique to parallel semantics. However, there is no need

to discuss the detailed structure of S at this stage.

These states are manipulated by elements of a domain C of "commands", onto which programs and parts of programs will be mapped by the semantic functions. The effect of any such $\theta \in C$ is to apply a sequence of indivisible primitive operations to a state σ. However, in the presence of other commands being executed in parallel with θ, the second and subsequent operations produced by θ will in general depend on what other commands do in between. Hence it will not do to have θ represented simply as a list of "primitive" state transformations, but rather as an object which, when applied to a state σ_0, gives a new state σ_1, transformed from σ_0 by a single primitive operation, together with another command which represents the rest of what θ has to do. The second command will later be applied to another state which will be σ_1 with (possibly) some operations from other commands applied to it. It is, of course, necessary to allow this process to stop. So C is made to include a special "termination" element θ_t. (cf Plotkin's [26] domain of resumptions where the result of applying a state is either another (final) state or an intermediate state and another resumption. But it is convenient in the present work to have the terminal command explicit, rather than have the situation where every command does at least one operation.) Also, to compare different interleavings of parallel commands, lists of operations are required rather than simply lists of transformed states. This leads to a domain C with structure:

$$C = [S \to [[S \to S] \times C]] + \{\theta_t\}$$

(note that $(\lambda\theta.\theta=\theta_t)$ is a continuous function).

The final state after executing a command θ from initial

state σ, in the absence of any simultaneous activity, is given by the function *Outcome*:

$$Outcome\theta\sigma =$$
$$(\theta=\theta_t \rightarrow \sigma,\ (\lambda\langle \chi,\theta'\rangle .Outcome\theta'(\chi\sigma))(\theta\sigma))$$

(χ will be used exclusively to denote elements of $[S\rightarrow S]$). Alternatively, the complete list of operations produced by θ can be found by means of the function *Expand*:

$$Expand\theta\sigma =$$
$$(\theta=\theta_t \rightarrow \langle\rangle,\ (\lambda\langle \chi,\theta'\rangle .\langle \chi\rangle \hat{}\ Expand\theta'(\chi\sigma))(\theta\sigma))$$

while the list of operations and the final result are related by:

$$Apply\chi^*\sigma =$$
$$(\#\chi^*=0 \rightarrow \sigma,\ Apply(\chi^*\dagger 1)(\chi_1\sigma)) .$$

This is satisfactory provided that χ^* is finite in length. However, if the program represented by θ does not terminate, the situation is more complicated: if χ^* is restricted to being a finite list, then the Expansion of a non-terminating command will simply be $\perp$; but this is not very helpful (though, of course, neither is a non-terminating command!) and there is no particular reason why χ^* should not be allowed to be infinite in length. The choice is not considered vital, since it is generally more profitable to be concerned with programs which at

least manage to complete something. Indications will, however, be given throughout this chapter of what needs to be done to cope with lists of infinite length.

Two useful relationships between the functions defined above are shown in the following obvious lemmas.

Lemma 1.1.1:

$Apply(Expand\theta\sigma)\sigma = Outcome\theta\sigma$.

Proof

by induction on $\nu=\#(Expand\theta\sigma)$.

If $\theta=\theta_t$

then $\mathcal{L}=Apply\langle\rangle\sigma=\sigma=\mathcal{R}$.

If $\theta\neq\theta_t$,

let $\langle\chi,\theta'\rangle=\theta\sigma$;

so $Expand\theta\sigma=\langle\chi\rangle\hat{}Expand\theta'(\chi\sigma)$;

so $\#(Expand\theta'(\chi\sigma))=\nu-1$;

so $\mathcal{L} = Apply(\langle\chi\rangle\hat{}Expand\theta'(\chi\sigma))\sigma$

$= Apply(Expand\theta'(\chi\sigma))(\chi\sigma)$

$= Outcome\theta'(\chi\sigma)$ by inductive hypothesis

$= Outcome\theta\sigma$

$= \mathcal{R}$.

(The case $\nu=\infty$ requires a simple application of fixed point induction (see [23]) using essentially the same inductive step.)

QED

Lemma 1.1.2:

$$Apply(\chi_1^* \hat{} \chi_2^*) = Apply\chi_2^* \circ Apply\chi_1^* \ .$$

Proof

by induction on $\nu = \#\chi_1^*$,

ie assuming true for $\chi_1'^*$ when $\#\chi_1'^* < \nu$,

show that $\forall\sigma$, $Apply(\chi_1^* \hat{} \chi_2^*)\sigma = Apply\chi_2^*(Apply\chi_1^*\sigma)$.

If $\nu = 0$

then $\mathcal{L} = Apply\chi_2^*\sigma$

and $\mathcal{R} = Apply\chi_2^*(Apply\langle\rangle\sigma) = Apply\chi_2^*\sigma$.

If $\nu > 0$ (but finite)

then $\mathcal{L} = Apply(\chi_1^* \hat{} \chi_2^*)\sigma$

$= Apply((\chi_1^* \hat{} \chi_2^*)\dagger 1)(((\chi_1^* \hat{} \chi_2^*)\downarrow 1)\sigma)$

$= Apply((\chi_1^* \dagger 1) \hat{} \chi_2^*)((\chi_1^* \downarrow 1)\sigma)$ since $\nu \geq 1$

$= Apply\chi_2^*(Apply(\chi_1^* \dagger 1)((\chi_1^* \downarrow 1)\sigma))$

by inductive hypothesis

$= Apply\chi_2^*(Apply\chi_1^*\sigma)$

$= \mathcal{R}$.

If $\nu = \infty$ then both sides are $\perp$.

QED

When several commands acting in parallel are considered, two more domains are needed. First, a domain P of "process identifiers" - a flat lattice of elements which are distinguishable but have no other special significance. They are used in lists of distinct elements, as index sets for lists of commands. Thus if $\theta_1, \ldots, \theta_\nu$ are ν elements of C, and π^* is list of ν distinct elements of P, a compound object $\theta^{\pi*}$ can be

created, such that, for $1 \le i \le \nu$, π_i indexes θ_i - this will be written simply as $\theta^{\pi^*}\pi_i = \theta_i$. This compound object will taken as an element of $[P^* \times [P \to C]]$, although $[P^* \times C^*]$ or $[P \times C]^*$ would do just as well. This formulation has the advantage that the index list π^* and the correspondence between each θ and its index are explicit, but is slightly clumsy in that the value of the second component is irrelevant for values of π not in the index list. However, such values will never be needed, and the above example will be written as $\theta^{\pi^*} = \langle \pi^*, \lambda\pi_i.\theta_i \rangle$. This notation will also be used for elements of the form $\chi^{*\pi^*} \in [P^* \times [P \to [S \to S]^*]]$.

Second, a domain Z of "schedules", whose elements are infinite lists of process identifiers. These will define the way in which parallel commands are interleaved, in a manner similar to Milne's prescriptive rosters [22]; thus the first element of a schedule which indexes a non-terminal command determines the next command for consideration, and this procedure is repeated using the tail of the original schedule, ie with elements up to and including the chosen process deleted. Schedules, of course, model the indeterminacy of a parallel computer caused by the variable relative timings of interactions. It must be emphasized that, in spite of the connotations which the word usually has, a schedule is not a device for imposing a certain pattern of operation, but rather a mathematical technique for describing what happens. The aim of this chapter is to show that, under certain conditions, a program is determinate, ie. its result is independent of the schedule(s) used. In order to ensure that anything waiting to be done does get done eventually, the following definition is needed:

Definition

A schedule ζ is *Fair* in π^* if

for every $\pi \triangleleft \pi^*$ and integer $\iota_0 > 0$

there is some $\iota \geq \iota_0$ s.t. $\zeta \downarrow \iota = \pi$

Hereafter, all schedules will be taken to be fair in the appropriate π^*.

The effect of executing the commands of an indexed list $\theta^{\pi*}$ in parallel, according to a schedule ζ (which is fair in π^*), and then continuing with a command θ, is given by the function *Combine*:

Definition

$$AreT\theta^{\pi*} =$$
$$(\forall \pi \triangleleft \pi^*)(\theta^{\pi*}\pi = \theta_t)$$

$$Choose\theta^{\pi*}\zeta =$$
$$(\lambda\pi.(\pi \triangleleft \pi^* \rightarrow \theta^{\pi*}\pi \neq \theta_t,\ False) \rightarrow \langle \pi, \zeta\dagger 1\rangle,\ Choose\theta^{\pi*}(\zeta\dagger 1))$$
$$(\zeta \downarrow 1)$$

$$Combine\theta^{\pi*}\zeta\theta =$$
$$(AreT\theta^{\pi*} \rightarrow \theta,\ \lambda\sigma.(\lambda\langle \pi, \zeta'\rangle.(\lambda\langle \chi, \theta'\rangle.\langle \chi, Combine\theta^{\pi*}[\theta'/\pi]\zeta'\theta\rangle)(\theta^{\pi*}\pi\sigma))$$
$$(Choose\theta^{\pi*}\zeta))$$

Here, the purpose of *Choose* is to pick out the first usable process π from a schedule, according to the criterion that the corresponding command is non-terminal (and, of course, that π is in the appropriate index list), assuming that there is such a

non-terminal command, ie unless $AreT\theta^{\pi*}$. The following lemma guarantees that *Choose* always has this effect.

Lemma 1.1.3:

If $\sim AreT\theta^{\pi*}$

then $Choose\theta^{\pi*}\zeta = \langle \zeta\downarrow\iota, \zeta\dagger\iota\rangle$

where ι is the least integer s.t. $\pi=\zeta\downarrow\iota$ satisfies

$\pi\lessdot\pi^*$ and $\theta^{\pi*}\pi\neq\theta_t$. (1)

Proof

Since $\sim AreT\theta^{\pi*}$,

there is at least one $\pi\lessdot\pi^*$ s.t. $\theta^{\pi*}\pi\neq\theta_{t_i}$

and by Fairness of ζ, $\exists\iota$ s.t. $\zeta\downarrow\iota=\pi$;

so proceed by induction on least ι s.t. $\zeta\downarrow\iota$ satisfies (1).

If $\iota=1$ then from definition of *Choose*,

$\mathcal{L}=\langle \zeta\downarrow 1, \zeta\dagger 1\rangle = \mathcal{R}$.

If $\iota>1$ then either $\sim(\zeta\downarrow 1\lessdot\pi^*)$ or $\theta^{\pi*}(\zeta\downarrow 1)=\theta_t$;

so $\mathcal{L}=Choose\theta^{\pi*}\zeta'$

where $\zeta'=\zeta\dagger 1$;

but least integer ι' s.t. $\zeta'\downarrow\iota'$ satisfies (1) is $\iota-1$;

so by inductive hypothesis,

$\mathcal{L} = Choose\theta^{\pi*}\zeta'$

$= \langle \zeta'\downarrow\iota', \zeta'\dagger\iota'\rangle$

$= \langle \zeta\downarrow\iota, \zeta\dagger\iota\rangle$.

QED

Having discovered how to combine a number of commands in parallel, little can be done without a method of keeping track of what one of them is doing. This is done by the function

Detail, which picks out the primitive operations done by process π from an Expansion of a Combination of $\theta^{\pi*}$.

Definition

$$Detail\theta^{\pi*}\zeta\pi\sigma =$$
$$(\theta^{\pi*}\pi=\theta_t \rightarrow \langle\rangle\ ,\ (\lambda\langle\pi',\zeta'\rangle.(\lambda\langle\chi,\theta'\rangle.(\pi=\pi' \rightarrow \langle\chi\rangle\ ,\ \langle\rangle)^{\frown}$$
$$Detail\theta^{\pi*}[\theta'/\pi']\zeta'\pi(\chi\sigma))$$
$$(\theta^{\pi*}\pi'\sigma))$$
$$(Choose\theta^{\pi*}\zeta))$$

The base for many later inductive proofs will depend on the fact that Expansions and Details have zero length precisely when the corresponding command is terminal. This is obvious for *Expand*, but slightly less so for *Detail*, so it merits another lemma:

Lemma 1.1.4:

If $\pi \lessdot \pi^*$

then $\#(Detail\theta^{\pi*}\zeta\pi\sigma)=0 \Leftrightarrow \theta^{\pi*}\pi=\theta_t$.

Proof

$\Leftarrow$ immediate from definition of *Detail*.

$\Rightarrow$ By Fairness of ζ, there is some ι s.t. $\zeta\downarrow\iota=\pi$.

Now proceed by induction on ι_0, the least such integer.

Suppose $\#(Detail\theta^{\pi*}\zeta\pi\sigma)=0$ and $\theta^{\pi*}\pi\neq\theta_t$

and let $\langle\pi',\zeta'\rangle = Choose\theta^{\pi*}\zeta$

and $\langle\chi,\theta'\rangle = \theta^{\pi*}\pi'\sigma$;

then $Detail\theta^{\pi*}\zeta\pi\sigma = (\pi=\pi' \rightarrow \langle\chi\rangle\ ,\langle\rangle)^{\frown}Detail\theta^{\pi*}[\theta'/\pi']\zeta'\pi(\chi\sigma)$;

but $\#(Detail\theta^{\pi*}\zeta\pi\sigma)=0$;

hence $\pi \neq \pi'$ and $\#(Detail\theta^{\pi *}[\theta'/\pi']\zeta'\pi(\chi\sigma))=0$;

but by 1.1.3,

$\langle \pi',\zeta' \rangle = \langle \zeta\downarrow\iota, \zeta\dagger\iota \rangle$ for some $\iota \leq \iota_0$;

but since $\pi \neq \pi'$, $\iota < i_0$;

hence $\zeta'\downarrow\iota' = \pi$ for $\iota' = \iota_0 - \iota < \iota_0$;

hence by inductive hypothesis, $\theta^{\pi *}[\theta'/\pi']\pi = \theta_t$;

ie $\theta^{\pi *}\pi = \theta_t$;

contradicting original assumption.

Hence $\#(Detail\theta^{\pi *}\zeta\pi\sigma)=0 \Rightarrow \theta^{\pi *}\pi = \theta_t$.

QED

Given the expansions produced by *Detail* for each process, they can be mixed together again, using the same schedule, with the function *Mix*, which works very much like *Combine*:

Definition

$Empty\chi^{*\pi *}$ =

$(\forall \pi \prec \pi^*)(\#\chi^{*\pi *}=0)$

$Select\chi^{*\pi *}\zeta$ =

$(\lambda\pi.(\pi \prec \pi^* \rightarrow \#\chi^{*\pi *} \neq 0,\ False) \rightarrow \langle \pi, \zeta\dagger 1 \rangle,\ Select\chi^{*\pi *}(\zeta\dagger 1))$

$(\zeta\downarrow 1)$

$Mix\chi^{*\pi *}\zeta$ =

$(Empty\chi^{*\pi *} \rightarrow \langle\rangle,\ (\lambda\langle \pi,\zeta' \rangle.\langle \chi^{*\pi *}\pi\downarrow 1 \rangle \hat{}\, Mix(\chi^{*\pi *}[\chi^{*\pi *}\pi\dagger 1/\pi])\zeta')$

$(Select\chi^{*\pi *}\zeta))$

Select and *Empty* have purposes precisely analogous to *Choose* and *AreT*, but acting on operation lists instead of lists of

commands. This suggests the following lemma precisely analogous to 1.1.3.

Lemma 1.1.5:

If $\sim Empty\chi^{*\pi*}$

then $Select\chi^{*\pi*} = \langle \zeta\downarrow\iota, \zeta\dagger\iota \rangle$

where ι is the least integer s.t. $\pi=\zeta\downarrow\iota$ satisfies

$\pi \lessdot \pi^*$ and $\#(\chi^{*\pi*}\pi) \neq 0$.

Proof

is so similar to that of 1.1.3 that it is omitted.

QED

Another function which will temporarily be useful gives the total length of a list of lists of operations.

Definition

$$LengthSum\chi^{*\pi*} = \sum_{\pi \lessdot \pi^*} \#\chi^{*\pi*}\pi$$

It features in the following lemma which demonstrates the rather obvious fact that the length of a Mixture of state transformation lists is independent of the schedule used.

Lemma 1.1.6:

$\#(Mix\chi^{*\pi*}\zeta) = LengthSum\chi^{*\pi*}$.

Proof

by induction on $\nu = LengthSum\chi^{*\pi*}$.

If $\nu=0$ then $Empty\chi^{*\pi*}$;

so $Mix\chi^{*\pi*}\zeta = \langle\rangle$;

ie $\ell = 0$.

If $\nu>0$ then $\sim Empty\chi^{*\pi*}$;

let $\langle \pi',\zeta' \rangle = Select\chi^{*\pi*}\zeta$

and $\chi'^{*\pi*}=\chi^{*\pi*}[\chi^{*\pi*}\pi'\dagger 1/\pi']$;

then $LengthSum\chi'^{*\pi*} = \sum_{\pi<\pi^*,\neq\pi'} \#\chi^{*\pi*}\pi + (\#\chi^{*\pi*}\pi'-1)$

$= (\sum_{\pi<\pi^*} \#\chi^{*\pi*}\pi)-1$

$= \nu-1$;

so by inductive hypothesis,

$\#(Mix\chi'^{*\pi*}\zeta')=\nu-1$;

so $\pounds = \#(\langle \chi^{*\pi*}\pi'\downarrow 1\rangle \hat{} Mix\chi'^{*\pi*}\zeta')$

$= 1+(\nu-1)$

$= \nu$;

The infinite case is easily proved by showing that, for any ν,

$LengthSum\chi^{*\pi*}\geq\nu \Rightarrow \#(Mix\chi^{*\pi*}\zeta)\geq\nu$.

QED

Not surprisingly, Combining commands together and Expanding the result gives the same answer as extracting their Details and then Mixing the resulting lists together with the same schedule; thus giving an important alternative formulation of the business of Combining parallel computations.

Theorem 1.1:

$Expand(Combine\theta^{\pi*}\zeta\theta_0)\sigma_0 =$

$Mix\langle \pi^*,\lambda\pi.Detail\theta^{\pi*}\zeta\pi\sigma_0\rangle\ \zeta\hat{}Expand\theta_0\sigma_1$

where $\sigma_1 = Outcome(Combine\theta^{\pi*}\zeta\theta_t)\sigma_0$.

Proof

by induction on $\nu = \#(Mix\langle \pi^*, \lambda\pi.Detail\theta^{\pi*}\zeta\pi\sigma_0\rangle\,\zeta)$.

First note that

$\nu=0 \Leftrightarrow \forall\pi\leq\pi^*,\ Detail\theta^{\pi*}\zeta\pi\sigma_0=\langle\rangle$ (1.1.6)

$\Leftrightarrow \forall\pi\leq\pi^*,\ \theta^{\pi*}\pi=\theta_t$ (1.1.4)

$\Leftrightarrow AreT\theta^{\pi*}$;

So if $\nu=0$ then $AreT\theta^{\pi*}$;

so $\mathcal{L}=Expand\theta_0\sigma_0$;

and $\mathcal{R}=Expand\theta_0\sigma_1=\mathcal{L}$

since $\sigma_1=Outcome\theta_t\sigma_0=\sigma_0$.

If $\nu\neq0$ then $\sim AreT\theta^{\pi*}$;

let $\langle\pi',\zeta'\rangle=Choose\theta^{\pi*}\zeta$

and ι_0 s.t. $\langle\pi',\zeta'\rangle=\langle\zeta{\downarrow}\iota_0,\zeta\dagger\iota_0\rangle$ (which exists by 1.1.3)

and $\langle\chi,\theta'\rangle=\theta^{\pi*}\pi'\sigma_0$

and $\chi^{*\pi*}=\langle\pi^*,\lambda\pi.Detail\theta^{\pi*}\zeta\pi\sigma_0\rangle$;

now for any $\pi\leq\pi^*$,

$\theta^{\pi*}\pi=\theta_t \Leftrightarrow \#(\chi^{*\pi*}\pi)=0$ (1.1.4) ;

so $\forall\iota<\iota_0$, since by 1.1.3, $\zeta{\downarrow}\iota\leq\pi^* \Rightarrow \theta^{\pi*}(\zeta{\downarrow}\iota)=\theta_t$,

$\zeta{\downarrow}\iota\leq\pi^* \Rightarrow \#(\chi^{*\pi*}(\zeta{\downarrow}\iota))=0$;

whereas $\zeta{\downarrow}\iota_0\leq\pi^*$ and $\theta^{\pi*}(\zeta{\downarrow}\iota_0)\neq\theta_t$;

so $\#(\chi^{*\pi*}(\zeta{\downarrow}\iota_0))\neq0$;

hence by 1.1.5,

$Select\chi^{*\pi*}\zeta=\langle\pi',\zeta'\rangle$;

so $\mathcal{L} = \langle\chi\rangle\,\hat{}\,Expand(Combine\theta^{\pi*}[\theta'/\pi']\zeta'\theta_0)(\chi\sigma)$;

and $\mathcal{R} = Mix\chi^{*\pi*}\zeta\,\hat{}\,\chi_0^*$

where $\chi_0^*=Expand\theta_0\sigma_1$,

$= \langle\chi^{*\pi*}\pi'{\downarrow}1\rangle\,\hat{}\,Mix\chi'^{*\pi*}\zeta'\,\hat{}\,\chi_0^*$

where $\chi'^{*\pi*}=\chi^{*\pi*}[\chi^{*\pi*}\pi'\dagger 1/\pi']$

$=\langle \pi^*, \lambda\pi.\pi=\pi' \rightarrow (Detail\theta^{\pi*}\zeta\pi'\sigma_0)\dagger 1,$

$Detail\theta^{\pi*}\zeta\pi\sigma_0\rangle$

$=\langle \pi^*, \lambda\pi.\pi=\pi' \rightarrow Detail\theta'^{\pi*}\zeta'\pi'(\chi\sigma_0),$

$Detail\theta^{\pi*}\zeta\pi\sigma_0\rangle$

where $\theta'^{\pi*}=\theta^{\pi*}[\theta'/\pi']$

$=\langle \pi^*, \lambda\pi.Detail\theta'^{\pi*}\zeta'\pi(\chi\sigma_0)\rangle$;

but $\#(Mix\langle \pi^*, \lambda\pi.Detail\theta'^{\pi*}\zeta'\pi(\chi\sigma_0)\rangle \zeta')$

$=\#((Mix\chi^{*\pi*}\zeta)\dagger 1)$

$=\nu-1$;

and also $\sigma_1=Outcome(Combine\theta^{\pi*}\zeta\theta_t)\sigma_0$

$=Outcome(Combine\theta'^{\pi*}\zeta'\theta_t)(\chi\sigma_0)$;

so by inductive hypothesis,

$\mathcal{R}=\langle \chi\rangle \hat{}\, Expand(Combine\theta'^{\pi*}\zeta'\theta_0)(\chi\sigma_0)$

$=\mathcal{L}$.

For the infinite case, the same inductive step is used to show that for any ν, $\#\mathcal{L}>\nu<\#\mathcal{R}$ and the first ν components of $\mathcal{L}$ and $\mathcal{R}$ are equal.

QED

1.2 Checking the Basic Operations

The significance of theorem 1.1 lies in the fact that ζ occurs only once on the LHS, but twice on the RHS (as well as the implicit occurrence in σ_1). This represents the fact that variations of schedule may make themselves felt in two ways. First, (corresponding to the occurrence as an argument of *Detail*,) the actual list of operations for any one process may

change since interacting processes may cause the intermediate states to be different. Second, interleaving the lists of operations in different ways can obviously produce different results if the operations are not commutative.

The first of these is dealt with by designing the semantics of any programming language under consideration so that the result of any given program is either the "right" one (ie, the unique one given by serial execution) or a special error value which will denoted by $\underline{?}$.

The function *Conts*, which is used in the semantic equations whenever the contents of a location in the store is required, provides a good example of how this is done. *Conts* requires two arguments - a location in the store, α, and an "expression continuation", κ, which takes a (stored) value as argument and gives an element of C. The result is another element of C. Assuming the existence of a primitive function *Fetch*$\alpha\sigma$ which finds the contents of a location α in the store part of the state σ, the simple definition of *Conts*, adequate for serial semantics, is:

$$Conts\alpha\kappa = \lambda\sigma.\kappa(Fetch\alpha\sigma)\sigma \ .$$

However, as described in chapter 0, *Conts* must indicate that it has communicated with the store, and yet it has made no change to it, so one might try:

$$Conts\alpha\kappa\sigma = \langle \lambda\sigma'.\sigma', \kappa(Fetch\alpha\sigma)\rangle \ .$$

But this gives no indication as to what part of the store was accessed, or why, nor does it satisfy the condition described above. So *Conts* is in fact defined as

$$Conts\alpha\kappa\sigma = \langle CheckFetch\alpha(Fetch\alpha\sigma), \kappa(Fetch\alpha\sigma)\rangle$$

$$\text{where } CheckFetch\alpha\beta = (\lambda\sigma'.Fetch\alpha\sigma' = \beta \rightarrow \sigma', \underline{?}) \ .$$

This means that if $Conts\alpha\kappa\sigma$ is applied to another state σ', the result depends only on whether the contents of α in σ' is the same as in σ. If so, then all is well and the program continues correctly; if not then *CheckFetch* ensures that the result $\underline{?}$ is immediately produced.

In general then, every (non-terminal) command will be required to satisfy a condition of the form:

for all $\sigma_1,\sigma_2 \in S$

either $\theta\sigma_1 = \theta\sigma_2$ or $(\theta\sigma_2\downarrow 1)\sigma_1 = \underline{?}$

and in either case $\theta\sigma_1\downarrow 2$ satisfies a similar condition;

and also, since this condition is not much use if the result is always $\underline{?}$:

$(\theta\sigma_1\downarrow 1)\sigma_1 \neq \underline{?}$.

Unfortunately this condition is recursive but not continuous, and so cannot be defined using *Fix*. However, the following form of definition is possible:

Definition

For each integer $\nu \geq 0$

$\theta \in C$ is ν-*Consistent* if one of the following holds:

i) $\nu = 0$;

ii) $\theta = \theta_t$;

iii) for all σ_1, σ_2, .

a) $(\theta\sigma_1\downarrow 2)$ is ν'-Consistent for all $\nu' < \nu$;

and b) $(\theta\sigma_1\downarrow 1)\sigma_1 \neq \underline{?}$ (provided $\sigma_1 \neq \underline{?}$) ;

and c) either $\theta\sigma_1 = \theta\sigma_2$ or $(\theta\sigma_2\downarrow 1)\sigma_1 = \underline{?}$.

θ is *Consistent* if it is ν-Consistent for all $\nu \geq 0$.

Note that if θ is ν_0-Consistent, then it is ν-Consistent for

$0 \leq \nu \leq \nu_0$. Furthermore, by the following lemma, this definition is adequate to describe the criterion mentioned above.

Lemma 1.2.1:

θ is Consistent iff

either $\theta = \theta_t$;

or for all σ_1, σ_2,

a) $(\theta\sigma_1 \downarrow 2)$ is Consistent;

and b) $(\theta\sigma_1 \downarrow 1)\sigma_1 \neq ?$ (provided $\sigma_1 \neq ?$) ;

and c) either $\theta\sigma_1 = \theta\sigma_2$ or $(\theta\sigma_2 \downarrow 1)\sigma_1 = ?$.

Proof

Trivial definition chasing.

QED

The definition is obviously extendable to indexed lists of commands, thus:

$\theta^{\pi *}$ is Consistent iff $\forall \pi \prec \pi^*$, $\theta^{\pi *}\pi$ is Consistent.

Two small lemmas concerned with Consistency are now required. The first expresses the fact that if the same schedule is used in both *Detail* and *Mix*, (which is equivalent by theorem 1.1 to using *Combine*,) then the result is never $?$ - a property which is inherited from, and is the only justification for (iii)(b) of Consistency. The second demonstrates that *Detail* produces a result very like *Expand* applied to a single command, except that the intermediate states in general vary out of the control of the process under consideration.

Lemma 1.2.2:

If $\theta^{\pi *}$ is Consistent

then $Outcome(Combine\theta^{\pi *}\zeta\theta_t)\sigma_0 \neq \underline{?}$

(provided $\sigma_0 \neq \underline{?}$).

Proof

by induction on $\nu = \#(Expand(Combine\theta^{\pi *}\zeta\theta_t)\sigma_0)$.

If $AreT\theta^{\pi *}$ then $Combine\theta^{\pi *}\zeta\theta_t = \theta_t$;

so $\ell = Outcome\,\theta_t\sigma_0 = \sigma_0 \neq \underline{?}$;

Otherwise,

let $\langle \pi, \zeta' \rangle = Choose\,\theta^{\pi *}\zeta$

and $\langle \chi, \theta' \rangle = \theta^{\pi *}\pi\sigma_0$;

then $\ell = Outcome(Combine\theta^{\pi *}[\theta'/\pi]\zeta'\theta_t)(\chi\sigma_0)$;

but $\#(Expand(Combine\theta^{\pi *}\zeta\theta_t)\sigma_0)$

$= \#(\langle \chi \rangle \hat{} Expand(Combine\theta^{\pi *}[\theta'/\pi]\zeta'\theta_t)(\chi\sigma_0))$

$- 1 + \#(Expand(Combine\theta^{\pi *}[\theta'/\pi]\zeta'\theta_t)(\chi\sigma_0))$;

and θ' is Consistent, so $\theta^{\pi *}[\theta'/\pi]$ is Consistent ;

and $\chi\sigma_0 \neq \underline{?}$;

so by inductive hypothesis

$Outcome(Combine\theta^{\pi *}[\theta'/\pi]\zeta'\theta_t)(\chi\sigma_0) \neq \underline{?}$.

(If $\nu = \infty$ then $\ell = \perp \neq \underline{?}$)

QED

Lemma 1.2.3:

Given $\theta \neq \theta_t$;

If $\theta_0^{\pi^*}$ is Consistent, $\theta_0^{\pi^*}\pi = \theta$, ζ_0 is fair in π^*, $\pi \prec \pi^*$

then $Detail\theta_0^{\pi^*}\zeta_0\pi\sigma_0 = \langle \theta\sigma_1\downarrow 1\rangle \hat{} Detail\theta_1^{\pi^*}\zeta_1\pi\sigma_2$

for some $\sigma_1, \sigma_2, \zeta_1, \theta_1^{\pi^*}$ satisfying

$\theta_1^{\pi^*}$ is Consistent, ζ_1 is fair in π^*, $\theta_1^{\pi^*}\pi = \theta\sigma_1\downarrow 2$.

Proof

Since ζ_0 is fair in π^*, $\exists \iota$ s.t. $\zeta_0\downarrow\iota = \pi$;

Proceed by induction on least such integer ι_0,

ie assume true for any ζ_0 s.t. $\zeta_0\downarrow\iota = \pi$ for some $\iota < \iota_0$;

by 1.1.3, $Choose\theta_0^{\pi^*}\zeta_0 = \langle \zeta_0\downarrow\iota, \zeta_0\dagger\iota\rangle$

for some $\iota \leq \iota_0$ since $\zeta_0\downarrow\iota_0 \prec \pi^*$ and $\theta_0^{\pi^*}(\zeta_0\downarrow\iota_0) \neq \theta_t$.

If $\iota = \iota_0$, ie $\zeta_0\downarrow\iota = \pi$

then $Detail\theta_0^{\pi^*}\zeta_0\pi\sigma_0$

$= \langle \theta\sigma_0\downarrow 1\rangle \hat{} Detail\theta_0^{\pi^*}[\theta\sigma_0\downarrow 2/\pi](\zeta_0\dagger\iota)\pi((\theta\sigma_0\downarrow 1)\sigma_0)$;

which is of the required form, with $\sigma_1 = \sigma_0$.

If $\iota < \iota_0$, then $\pi' = \zeta_0\downarrow\iota \neq \pi$;

so $Detail\theta_0^{\pi^*}\zeta_0\pi\sigma_0 = Detail\theta_2^{\pi^*}\zeta_0'\pi\sigma_3$

where $\theta_2^{\pi^*} = \theta_0^{\pi^*}[\theta_0^{\pi^*}\pi'\sigma_0\downarrow 2/\pi']$ which is Consistent

and $\zeta_0' = \zeta_0\dagger\iota$

and $\sigma_3 = (\theta_0^{\pi^*}\pi'\sigma_0\downarrow 1)\sigma_0$;

so $\theta_2^{\pi^*}\pi = \theta$, ζ_0' is fair in π^*,

and $\zeta_0'\downarrow\iota' = \pi$ where $\iota' = \iota_0 - \iota < \iota_0$;

hence by inductive hypothesis,

$\mathcal{L} = \langle \theta\sigma_1\downarrow 1\rangle \hat{} Detail\theta_1^{\pi^*}\zeta_1\pi\sigma_2$.

QED

This leads to the essential consequence of Consistency -

that only one "correct" (ie not $\underline{?}$) answer can be obtained by varying the schedule argument of *Detail*.

Theorem 1.2:

For fixed ζ, σ, and Consistent θ^{π^*},

as $\theta_\pi^{\pi^*}$, ζ_π, σ_π vary (for each $\pi \lessdot \pi^*$) s.t.

for each $\pi \lessdot \pi^*$, ζ_π is fair in π^*, $\theta_\pi^{\pi^*}\pi = \theta^{\pi^*}\pi$

and $\theta_\pi^{\pi^*}$ Consistent,

there is (at most) one value of $\chi^{*\pi^*} = \langle \pi^*, \lambda\pi.\mathit{Detail}\,\theta_\pi^{\pi^*}\zeta_\pi\pi\sigma_\pi\rangle$

and one corresponding value of $\chi^* = \mathit{Mix}\chi^{*\pi^*}\zeta$

for which $\mathit{Apply}\chi^*\sigma \neq \underline{?}$.

Hence as ζ_1 varies, there is at most one value of

$\chi^* = \mathit{Mix}\langle \pi^*, \lambda\pi.\mathit{Detail}\,\theta^{\pi^*}\zeta_1\pi\sigma\rangle\,\zeta_2$

for which $\mathit{Apply}\chi^*\sigma \neq \underline{?}$.

Proof

For $\iota = 1,2$, let $\theta_{\pi\iota}^{\pi^*}, \zeta_{\pi\iota}, \sigma_{\pi\iota}$ satisfy the hypotheses

and let $\chi_\iota^{*\pi^*} = \langle \pi^*, \lambda\pi.\mathit{Detail}\,\theta_{\pi\iota}^{\pi^*}\zeta_{\pi\iota}\pi\sigma_{\pi\iota}\rangle$

and $\chi_\iota^* = \mathit{Mix}\chi_\iota^{*\pi^*}\zeta$;

Then use induction on $\nu = \mathit{Min}(\#\chi_1^*, \#\chi_2^*)$,

ie assume that

if $\#\chi_1^* < \nu$ and $\#\chi_2^* < \nu$

then $\chi_1^{*\pi^*} = \chi_2^{*\pi^*}$ or $\mathit{Apply}\chi_1^*\sigma = \underline{?}$ or $\mathit{Apply}\chi_2^*\sigma = \underline{?}$.

First, note that for each $\pi \lessdot \pi^*$, and $\iota = 1,2$,

$$\#(\chi_\iota^{*\pi^*}\pi) = 0 \Leftrightarrow \#(\mathit{Detail}\,\theta_{\pi\iota}^{\pi^*}\zeta_{\pi\iota}\pi\sigma_{\pi\iota}) = 0$$

$$\Leftrightarrow \theta_{\pi\iota}^{\pi^*}\pi = \theta_t \quad \text{(by 1.1.4)}$$

$$\Leftrightarrow \theta^{\pi^*}\pi = \theta_t \ ; \qquad (1)$$

Hence $\#\chi_1^* = 0 \Leftrightarrow (\forall \pi \lessdot \pi^*, \#(\chi_1^{*\pi^*}\pi) = 0)$ (by 1.1.6)

$$\Leftrightarrow (\forall \pi \lessdot \pi^*, \#(\chi_2^{*\pi^*}\pi) = 0)$$

$$\Leftrightarrow \#\chi_2^* = 0 \ ;$$

so if $\#\chi_1^*=0$ result is trivially true;

so assume $\#\chi_1^*\neq 0\neq\#\chi_2^*$;

then $\sim AreT\theta^{\pi*}$ and $\sim Empty\chi_1^{*\pi*}$ and $\sim Empty\chi_2^{*\pi*}$;

now let $\langle \pi',\zeta'\rangle = Select\chi_1^{*\pi*}\zeta$;

but from 1.1.5 and (1),

$\langle \pi',\zeta'\rangle = Select\chi_2^{*\pi*}\zeta$;

so $\chi_1^* = Mix\chi_1^{*\pi*}\zeta$

$= \langle \chi_1^{*\pi*}\pi'\downarrow 1\rangle \hat{} Mix\chi_1'^{*\pi*}\zeta'$

where $\chi_1'^{*\pi*}=\chi_1^{*\pi*}[\chi_1^{*\pi*}\pi'\dagger 1/\pi']$;

but $\chi_1^{*\pi*}\pi' = Detail\theta^{\pi*}_{\pi'_1}\zeta_{\pi'_1}\pi'\sigma_{\pi'_1}$

$= \langle \theta^{\pi*}\pi'\sigma_1\downarrow 1\rangle \hat{} Detail\theta_1^{\pi*}\zeta_1\pi'\sigma_1'$

for some $\sigma_1,\sigma_1',\theta_1^{\pi*},\zeta_1$ satisfying

$\theta_1^{\pi*}$ is Consistent, ζ_1 fair in π^*,

and $\theta_1^{\pi*}\pi'=\theta^{\pi*}\pi'\sigma_1\downarrow 2$

(by 1.2.3);

Hence $\chi_1'^{*\pi*}=\langle \pi^*,\lambda\pi.Detail\theta'^{\pi*}_{\pi 1}\zeta'_{\pi 1}\pi\sigma'_{\pi 1}\rangle$

where $\theta'^{\pi*}_{\pi 1}\pi=\theta_1'^{\pi*}\pi$

where $\theta_1'^{\pi*}=\langle \pi^*,\lambda\pi.\pi=\pi'\rightarrow\theta^{\pi*}\pi\sigma_1\downarrow 2,\theta^{\pi*}\pi\rangle$;

and $\chi_1^*\downarrow 1=\theta^{\pi*}\pi'\sigma_1\downarrow 1=\theta_0\sigma_1\downarrow 1$ say .

If $\theta_0\sigma_1=\theta_0\sigma_2$

then $\chi_1^*\downarrow 1=\chi_2^*\downarrow 1$

and $\theta_1'^{\pi*}=\theta_2'^{\pi*}$;

so by inductive hypothesis,

(since $\chi_1'^{*}=Mix\chi_1'^{*\pi*}\zeta'=\chi_1^*\dagger 1$ has length 1 less than χ_1^*) ,

either $\chi_1'^{*\pi*}=\chi_2'^{*\pi*}$;

ie for $\pi\neq\pi'$,

$\chi_1^{*\pi*}\pi=\chi_1'^{*\pi*}\pi=\chi_2'^{*\pi*}\pi=\chi_2^{*\pi*}\pi$

and $\chi_1^{*\pi*}\pi'=\langle \chi_1^*\downarrow 1\rangle \hat{}\chi_1'^{*\pi*}\pi'=\langle \chi_2^*\downarrow 1\rangle \hat{}\chi_2'^{*\pi*}\pi'=\chi_2^{*\pi*}\pi'$;

ie $\chi_1^{*\pi*}=\chi_2^{*\pi*}$;

or $Apply\chi_1^*\sigma = Apply\chi_1'^*((\chi_1^*\downarrow 1)\sigma) = ?$;

or $Apply\chi_2^*\sigma = Apply\chi_2'^*((\chi_2^*\downarrow 1)\sigma) = ?$.

If $\theta_0\sigma_1 \neq \theta_0\sigma_2$

then for at least one of $\iota=1,2$, $\theta_0\sigma_\iota \neq \theta_0\sigma$;

so by Consistency, $(\theta_0\sigma_\iota\downarrow 1)\sigma = ?$;

ie $(\chi_\iota^*\downarrow 1)\sigma = ?$;

hence $Apply\chi_\iota^*\sigma = ?$.

(If one of χ_1^*, χ_2^* has infinite length, then this proof still holds, since ν remains finite; if both have infinite length, so $\nu=\infty$, then the proof has to be modified as in 1.1.)

QED

1.3 Mixing the Basic Operations

Turning now to the ways in which different interleavings of the primitive operations affect the result, a concept of equivalence of χ lists will be introduced which, while not insisting on equality of differently interleaved lists, will nevertheless ensure that the same result is always obtained; then a simple condition on the individual operations will ensure that this equivalence holds.

First, though, two functions are needed to examine the workings of *Mix* in more detail. They are:

$$MixMap\chi^{*\pi*}\zeta\langle\pi,\iota\rangle =$$
$$(\lambda\langle\pi',\zeta'\rangle.(\pi=\pi'\wedge\iota=1)\rightarrow 1,$$
$$1+MixMap(\chi^{*\pi*}[\chi^{*\pi*}\pi'\dagger 1/\pi'])\zeta'\langle\pi,(\pi=\pi'\rightarrow\iota-1,\iota)\rangle)$$
$$(Select\chi^{*\pi*}\zeta)$$

$$UnMixMap\chi^{*\pi*}\zeta\iota =$$
$$(\lambda\langle\pi,\zeta'\rangle.\iota=1\rightarrow\langle\pi,1\rangle,$$
$$(\lambda\langle\pi',\iota'\rangle.\langle\pi',\pi=\pi'\rightarrow\iota'+1,\iota'\rangle)$$
$$(UnMixMap(\chi^{*\pi*}[\chi^{*\pi*}\pi\dagger 1/\pi])\zeta'(\iota-1)))$$
$$(Select\chi^{*\pi*}\zeta)$$

These provide a mapping between $\langle\pi,\iota\rangle$ index pairs for lists of lists of χ's and indexes for the corresponding Mixed lists, as shown by the following two lemmas.

Lemma 1.3.1:

If $\pi\prec\pi^*$, $1\leq\iota_0\leq\#\chi^{*\pi*}\pi$

and $\chi^* = Mix\chi^{*\pi*}\zeta$

and $\iota_1 = MixMap\chi^{*\pi*}\zeta\langle\pi,\iota_0\rangle$

then i) $1\leq\iota_1\leq\#\chi^*$;

ii) $\langle\pi,\iota_0\rangle = UnMixMap\chi^{*\pi*}\zeta\iota_1$;

iii) $\chi^*\downarrow\iota_1 = \chi^{*\pi*}\pi\downarrow\iota_0$.

Proof

If $Empty\chi^{*\pi*}$, then no such ι_0 exists;

so assume $\sim Empty\chi^{*\pi*}$;

let $\langle\pi',\zeta'\rangle = Select\chi^{*\pi*}\zeta$;

and proceed by induction on ι_1 .

Case 1: $\pi=\pi'$ and $\iota_0=1$:

Then by definition of *MixMap*, $\iota_1=1$;

so i) $1\leq\iota_1\leq\#\chi^*$ since by 1.1.6, $\#\chi^* = \sum_{\pi\in\pi^*} \chi^{*\pi^*}\pi > 0$;

ii) $\mathit{UnMixMap}\chi^{*\pi^*}\zeta\iota_1 = \mathit{UnMixMap}\chi^{*\pi^*}\zeta 1$

$= \langle \pi', 1\rangle$

$= \langle \pi, \iota_0\rangle$;

iii) $\chi^*\downarrow\iota_1 = \chi^*\downarrow 1$

$= \chi^{*\pi^*}\pi'\downarrow 1$

$= \chi^{*\pi^*}\pi\downarrow\iota_0$.

Case 2: $\pi\neq\pi'$ or $\iota_0>1$:

let $\chi'^{*\pi^*}=\chi^{*\pi^*}[\chi^{*\pi^*}\pi'\dagger 1/\pi']$

and $\chi'^*=\mathit{Mix}\chi'^{*\pi^*}\zeta'=\chi^*\dagger 1$

and let $\iota_0'=(\pi=\pi'\rightarrow\iota_0-1,\iota_0)$

and $\iota_1'=\mathit{MixMap}\chi'^{*\pi^*}\zeta'\langle\pi,\iota_0'\rangle$;

then from definition of *MixMap*, $\iota_1=\iota_1'+1$;

Now $\forall\pi_0\in\pi^*$, $\#\chi'^{*\pi^*}\pi_0=(\pi_0=\pi'\rightarrow\#(\chi^{*\pi^*}\pi_0)-1,\#(\chi^{*\pi^*}\pi_0))$;

and if $\pi=\pi'$, $\iota_0>1$;

so $1\leq\iota_0'\leq\#(\chi'^{*\pi^*}\pi)$;

and also $\#\chi'^*=\#\chi^*-1$;

hence by inductive hypothesis,

i) $1\leq\iota_1'\leq\#\chi'^*$;

so $1\leq 2\leq\iota_1\leq\#\chi^*$;

ii) $\langle\pi,\iota_0'\rangle = \mathit{UnMixMap}\chi'^{*\pi^*}\zeta'\iota_1'$

$= \mathit{UnmixMap}\chi'^{*\pi^*}\zeta'(\iota_1-1)$;

so $\mathit{UnMixMap}\chi^{*\pi^*}\zeta\iota_1 = \langle\pi,\pi=\pi'\rightarrow\iota_0'+1,\iota_0'\rangle$

$= \langle\pi,\iota_0\rangle$;

iii) $\chi'^*\downarrow\iota_1' = \chi'^{*\pi^*}\pi\downarrow\iota_0'$;

so $\chi^*\downarrow\iota_1 = \chi^*\downarrow(\iota_1'+1)$

$$= \chi'^{*}\downarrow \iota'_1$$
$$= \chi'^{*\pi^{*}}\pi\downarrow \iota'_0$$
$$= \chi^{*\pi^{*}}\pi\downarrow \iota_0 \ .$$

QED

Lemma 1.3.2:

If $\chi^{*} = Mix\chi^{*\pi^{*}}\zeta$

and $1 \leq \iota_1 \leq \#\chi^{*}$

and $\langle \pi, \iota_0 \rangle = UnMixMap\chi^{*\pi^{*}}\zeta\iota_1$

then i) $\pi \lessdot \pi^{*}$ and $1 \leq \iota_0 \leq \#\chi^{*\pi^{*}}\pi$;

ii) $\iota_1 = MixMap\chi^{*\pi^{*}}\zeta\langle \pi, \iota_0 \rangle$;

iii) $\chi^{*}\downarrow \iota_1 = \chi^{*\pi^{*}}\pi\downarrow \iota_0$.

Proof

This can be proved in a similar way to 1.3.1, but it is simpler in the finite case to observe that, considered as a mapping from the set $\{\langle \pi, \iota_0 \rangle : \pi \lessdot \pi^{*} \wedge 1 \leq \iota_0 \leq \#\chi^{*\pi^{*}}\pi\}$ to $\{\iota_1 : 1 \leq \iota_1 \leq \#\chi^{*}\}$, $MixMap\chi^{*\pi^{*}}\zeta$ is injective (since by 1.3.1, $UnMixMap\chi^{*\pi^{*}}\zeta$ is its inverse); however, since by 1.1.6, these sets are equinumerous, this implies that $MixMap\chi^{*\pi^{*}}\zeta$ is bijective.

Hence ι_1 is the image of an element in the first set, and by 1.3.1(ii), this element is precisely $\langle \pi, \iota_0 \rangle$.

Hence (i), (ii) and (iii) follow from 1.3.1(iii).

QED

The next lemma shows that *Mix* preserves the order of operations within any one process.

Lemma 1.3.3:

If $\pi \lessdot \pi^*$ and $1 \le \iota_1, \iota_2 \le \#(\chi^{*\pi^*}\pi)$

then $\iota_1 < \iota_2 \Leftrightarrow MixMap\chi^{*\pi^*}\zeta\langle \pi, \iota_1 \rangle < MixMap\chi^{*\pi^*}\zeta\langle \pi, \iota_2 \rangle$.

Proof

Assume $\sim Empty\chi^{*\pi^*}$, since otherwise there is nothing to prove

and use induction on ι_2.

let $\phi = MixMap\chi^{*\pi^*}\zeta$

and $\psi = UnMixMap\chi^{*\pi^*}\zeta$;

let $\langle \pi', \zeta' \rangle = Select\chi^{*\pi^*}\zeta$

and $\chi'^{*\pi^*} = \chi^{*\pi^*}[\chi^{*\pi^*}\pi' \dagger 1/\pi']$

and $\phi' = MixMap\chi'^{*\pi^*}\zeta'$

and $\psi' = UnMixMap\chi'^{*\pi^*}\zeta'$.

Case 1: $\pi = \pi'$ and $\iota_1 = 1$:

then $\phi\langle \pi, \iota_1 \rangle = 1$;

so $\iota_1 < \iota_2 \Rightarrow \iota_2 > 1$

$\Rightarrow \phi\langle \pi, \iota_2 \rangle = 1 + \phi'\langle \pi, (\pi = \pi' \rightarrow \iota_2 - 1, \iota_2) \rangle$

≥ 2 by 1.3.1(i)

$\Rightarrow \phi\langle \pi, \iota_2 \rangle > 1$

$\Rightarrow \phi\langle \pi, \iota_1 \rangle < \phi\langle \pi, \iota_2 \rangle$;

and $\phi\langle \pi, \iota_1 \rangle < \phi\langle \pi, \iota_2 \rangle \Rightarrow \phi\langle \pi, \iota_2 \rangle > 1$

$\Rightarrow \iota_2 \neq 1$ since $\pi = \pi'$

$\Rightarrow \iota_2 > \iota_1$.

Case 2: $\pi = \pi'$ and $\iota_1 > 1$:

if $\iota_1 < \iota_2$

then $\phi\langle \pi, \iota_1 \rangle = 1 + \phi'\langle \pi, \iota_1 - 1 \rangle$

$< 1 + \phi'\langle \pi, \iota_2 - 1 \rangle$ by inductive hypothesis

$$< \phi\langle \pi, \iota_2 \rangle \ ;$$

and conversely,

$$\phi\langle \pi, \iota_1 \rangle < \phi\langle \pi, \iota_2 \rangle \Rightarrow 1+\phi'\langle \pi, \iota_1 - 1 \rangle < 1+\phi'\langle \pi, \iota_2 - 1 \rangle$$
$$\Rightarrow \phi'\langle \pi, \iota_1 - 1 \rangle < \phi'\langle \pi, \iota_2 - 1 \rangle$$
$$\Rightarrow \iota_1 - 1 < \iota_2 - 1 \quad \text{by inductive hypothesis}$$
$$\Rightarrow \iota_1 < \iota_2 \ ;$$

Case 3: $\pi \neq \pi'$:

$$\iota_1 < \iota_2 \Rightarrow \phi\langle \pi, \iota_1 \rangle = 1+\phi'\langle \pi, \iota_1 \rangle < 1+\phi'\langle \pi, \iota_2 \rangle = \phi\langle \pi, \iota_2 \rangle$$
$$\phi\langle \pi, \iota_1 \rangle < \phi\langle \pi, \iota_2 \rangle \Rightarrow 1+\phi'\langle \pi, \iota_1 \rangle < 1+\phi'\langle \pi, \iota_2 \rangle$$
$$\Rightarrow \iota_1 < \iota_2 \ .$$

QED

When lists of primitive operations are interleaved in different ways, it is obviously too much to expect the resulting lists to be identical, but it will be enough to insist that the results of *Apply*ing them are the same. Furthermore, the resulting lists will, of course, contain the same operations, though not in the same order. This prompts the definition of equivalence of χ lists:

Definition $\chi_1^* \approx \chi_2^*$

iff χ_1^* is a permutation of χ_2^*

and $Apply\chi_1^* = Apply\chi_2^*$.

It is obvious that $\approx$ is an equivalence relation.

To guarantee that this equivalence holds, conditions need to be imposed upon the lists to be Mixed.

Definition

$\chi^{*\pi*}$ is *Miscible* if

$\forall \pi_1, \pi_2 \lessdot \pi^*$ s.t. $\pi_1 \neq \pi_2$

and $\forall \iota_1, \iota_2$ s.t. $1 \leq \iota_1 \leq \#(\chi^{*\pi*}\pi_1)$ and $1 \leq \iota_2 \leq \#(\chi^{*\pi*}\pi_2)$

$\chi^{*\pi*}\pi_1 \downarrow \iota_1$ and $\chi^{*\pi*}\pi_2 \downarrow \iota_2$ commute.

The following theorem shows that this condition is indeed sufficient.

Theorem 1.3:

If $\chi^{*\pi*}$ is Miscible

then for any ζ, ζ' fair in π^*,

$$Mix\chi^{*\pi*}\zeta \approx Mix\chi^{*\pi*}\zeta' .$$

Proof

let $\phi = MixMap\chi^{*\pi*}\zeta$

and $\psi = UnMixMap\chi^{*\pi*}\zeta$

and $\chi^* = Mix\chi^{*\pi*}\zeta$

and similarly for ϕ', ψ', χ'^* using ζ' instead of ζ ;

let $N = \#\chi'^*$;

for $\nu = 0, 1, \ldots, N$, ϕ_ν and ψ_ν will be defined satisfying:

i) $\pi \lessdot \pi^* \wedge 1 \leq \iota_0 \leq \#\chi^{*\pi*}\pi \wedge \iota_1 = \phi_\nu\langle \pi, \iota_0 \rangle$

$\Leftrightarrow 1 \leq \iota_1 \leq N \wedge \langle \pi, \iota_0 \rangle = \psi_\nu \iota_1$;

ii) If $\pi \lessdot \pi^*$ and $1 \leq \iota_1, \iota_2 \leq \#\chi^{*\pi*}\pi$

then $\iota_1 < \iota_2 \Leftrightarrow \phi_\nu\langle \pi, \iota_1 \rangle < \phi_\nu\langle \pi, \iota_2 \rangle$;

iii) $\forall \iota \leq \nu$, $\psi_\nu \iota = \psi' \iota$;

iv) If χ^*_ν is given by

$\#\chi^*_\nu = N$ and

for $1 \leq \iota_1 \leq N$, $\chi^*_\nu \downarrow \iota_1 = \chi^{*\pi*}\pi \downarrow \iota_0$

where $\langle \pi, \iota_0 \rangle = \psi_\nu \iota_1$;

then $\chi^*_\nu \approx \chi^*$.

Taking $\phi_0=\phi$ and $\psi_0=\psi$,

(i) is satisfied (by 1.3.1 and 1.3.2);

(ii) is satisfied (by 1.3.3);

(iii) is vacuously true;

(iv) is true since $\chi^*_0=\chi^*$ (by 1.1.6, 1.3.1, 1.3.2);

for $1\le\nu\le N$, assume $\phi_{\nu-1},\psi_{\nu-1}$ have been constructed satisfying (i) - (iv);

let $\langle \pi',\iota'\rangle = \psi'\nu$

and $\nu' = \phi_{\nu-1}\langle \pi',\iota'\rangle$;

Now consider any $\iota\le N$ s.t. $\phi_{\nu-1}(\psi'\iota)\le\nu-1$;

then by (i) and (iii),

$$\psi'(\phi_{\nu-1}(\psi'\iota)) = \psi_{\nu-1}(\phi_{\nu-1}(\psi'\iota)) = \psi'\iota \ ;$$

so $$\begin{aligned}\iota &= \phi'(\psi'\iota)\\ &= \phi'(\psi'(\phi_{\nu-1}(\psi'\iota)))\\ &= \phi_{\nu-1}(\psi'\iota)\\ &< \nu \ ;\end{aligned}$$

so if $\iota\ge\nu$ then $\phi_{\nu-1}(\psi'\iota)\ge\nu$; (1)

and similarly $\phi'(\psi_{\nu-1}\iota)\ge\nu$;

in particular, $\nu' = \phi_{\nu-1}(\psi'\nu)\ge\nu$;

Furthermore, suppose $\exists\nu''$ s.t.

$\nu\le\nu''<\nu'$ and $\psi_{\nu-1}\nu''\downarrow 1=\pi'$;

then by (ii), since $\psi_{\nu-1}\nu'\downarrow 1=\psi_{\nu-1}\nu''\downarrow 1=\pi'$,

$\psi_{\nu-1}\nu''\downarrow 2<\psi_{\nu-1}\nu'\downarrow 2$;

so by 1.3.3, $\phi'(\psi_{\nu-1}\nu'')<\phi'(\psi_{\nu-1}\nu')$;

but then by (1),

$\nu\le\phi'(\psi_{\nu-1}\nu'')<\phi'(\psi_{\nu-1}\nu')=\nu$;

which is a contradiction.

Hence if $\nu \leq \nu'' < \nu'$

then $\psi_{\nu-1}\nu''\downarrow 1 \neq \pi'$; (2)

so by Miscibility of $\chi^{*\pi*}$ and definition of $\chi^*_{\nu-1}$,

$\chi^*_{\nu-1}\downarrow\nu'$ and $\chi^*_{\nu-1}\downarrow\nu''$ commute ;

So let $\psi_\nu = \lambda\iota.(\iota=\nu \to \psi_{\nu-1}\nu',$

$\nu<\iota\leq\nu' \to \psi_{\nu-1}(\iota-1),$

$\psi_{\nu-1}\iota)$

and $\phi_\nu = \lambda\langle\pi,\iota\rangle.(\phi_{\nu-1}\langle\pi,\iota\rangle=\nu' \to \nu,$

$\nu\leq\phi_{\nu-1}\langle\pi,\iota\rangle<\nu' \to \phi_{\nu-1}\langle\pi,\iota\rangle+1,$

$\phi_{\nu-1}\langle\pi,\iota\rangle)$;

Then:

(i) follows from the definition of ϕ_ν and ψ_ν

and from the corresponding result for $\phi_{\nu-1}$ and $\psi_{\nu-1}$;

(ii) follows similarly except in the cases where

$\phi_{\nu-1}\langle\pi,\iota_1\rangle=\nu'$ and $\nu\leq\phi_{\nu-1}\langle\pi,\iota_2\rangle<\nu'$

or with ι_1 and ι_2 interchanged ;

but these cases are impossible by (2) ;

(iii) for $\iota<\nu$, $\psi_\nu\iota=\psi_{\nu-1}\iota=\psi'\iota$;

and for $\iota=\nu$, $\psi_\nu\iota=\langle\pi',\iota'\rangle=\psi'\nu$;

(iv) for $1\leq\iota_1\leq N$,

$\chi^*_\nu\downarrow\iota_1=\chi^*_{\nu-1}\downarrow\iota_2$

where $\iota_2=\phi_{\nu-1}(\psi_\nu\iota_1)$

$=(\iota_1=\nu \to \nu',\ \nu<\iota_1\leq\nu' \to \iota_1-1,\ \iota_1)$;

so χ^*_ν is a permutation of $\chi^*_{\nu-1}$;

and if χ^*_α denotes first $\nu-1$ components of $\chi^*_{\nu-1}$

and χ^*_β denotes components ν to $\nu'-1$ of $\chi^*_{\nu-1}$

and $\chi=\chi^*_{\nu-1}\downarrow\nu'$

and $\chi^*_\gamma=\chi^*_{\nu-1}\dagger\nu'$

so that $\chi^*_{\nu-1}=\chi^*_\alpha\hat{}\chi^*_\beta\hat{}\langle\chi\rangle\hat{}\chi^*_\gamma$,

then $\chi^*_\nu = \chi^*_\alpha \hat{} \langle \chi \rangle \hat{} \chi^*_\beta \hat{} \chi^*_\gamma$;

but χ commutes with each component of χ^*_β ;

so (using 1.1.2) $Apply\chi^*_\beta \circ \chi = \chi \circ Apply\chi^*_\beta$;

so $Apply\chi^*_\nu = Apply\chi^*_\gamma \circ Apply\chi^*_\beta \circ \chi \circ Apply\chi^*_\alpha$

$= Apply\chi^*_\gamma \circ \chi \circ Apply\chi^*_\beta \circ Apply\chi^*_\alpha$

$= Apply\chi^*_{\nu-1}$;

so $\chi^*_\nu \approx \chi^*_{\nu-1} \approx \chi^*$;

But from (iii) and (iv), $\chi^*_N = \chi'^*$;

Hence $\chi'^* \approx \chi^*$.

For the infinite case, $Apply\chi^* = \perp$ since $\#\chi^* = \infty$, and the permutation is given by *MixMap* and *UnMixMap* in the same way as in the finite case.

QED

1.4 The Conditions for Equivalence

Having examined separately the various effects of using different schedules, the pieces can now be put together, and parallel execution compared with sequential. The aim of this section is to summarise the (semantic) conditions for equivalence of serial and parallel computation.

First, the results of the last three sections can be combined into the following lemma:

Lemma 1.4.1:

If $\theta^{\pi *}$ and θ_0 are Consistent (and $\sigma_0 \neq ?$)
and $\langle \pi^*, \lambda\pi.Detail\theta^{\pi *}\zeta\pi\sigma\rangle$ is Miscible for all ζ fair in π^*
and if ζ_1, ζ_2 are fair in π^*
then $Expand(Combine\theta^{\pi *}\zeta_1\theta_0)\sigma_0 \approx Expand(Combine\theta^{\pi *}\zeta_2\theta_0)\sigma_0$.

Proof

For $\iota=1,2$,

let $\chi_\iota^{*\pi *}=\langle \pi^*, \lambda\pi.Detail\theta^{\pi *}\zeta_\iota\pi\sigma\rangle$
and $\chi_\iota^*=Mix\chi_\iota^{*\pi *}\zeta_\iota$
and let $\chi_3^*=Mix\chi_1^{*\pi *}\zeta_2$;

Consider first the case $\theta_0=\theta_t$;
then $\mathcal{L}=\chi_1^*$ and $\mathcal{R}=\chi_2^*$ (by 1.1);
but by 1.3,

$$\chi_1^*=Mix\chi_1^{*\pi *}\zeta_1 \approx Mix\chi_1^{*\pi *}\zeta_2=\chi_3^* \quad ; \qquad (1)$$

but by 1.2,

either $\chi_2^*=\chi_3^*$
or $Apply\chi_2^*\sigma_0 \neq ?$
or $Apply\chi_3^*\sigma_0 \neq ?$;

but $Apply\chi_2^*\sigma_0 = Outcome(Combine\theta^{\pi *}\zeta_2\theta_t)\sigma_0$ (by 1.1.1)
$\neq ?$ (by 1.2.2);
and $Apply\chi_3^*\sigma_0 = Apply\chi_1^*\sigma_0$ (by (1))
$\neq ?$ (similarly);

so $\chi_2^*=\chi_3^*$;
hence $\chi_1^* \approx \chi_2^*$;

Now consider the general case:

For $\iota=1,2$,

let $\sigma_\iota=Apply\chi_\iota^*\sigma_0$
$=Outcome(Combine\theta^{\pi *}\zeta_\iota\theta_t)\sigma_0$;

then from above, $\sigma_1=\sigma_2$;

ie $Expand\theta_0\sigma_1=Expand\theta_0\sigma_2$;

but (using 1.1 twice),

$$Expand(Combine\theta^{\pi*}\zeta_\iota\theta_0)\sigma_0 = Expand(Combine\theta^{\pi*}\zeta_\iota\theta_t)\sigma_0 \hat{} Expand\theta_0\sigma_\iota ;$$

and $Expand(Combine\theta^{\pi*}\zeta_1\theta_t)\sigma_0 \approx Expand(Combine\theta^{\pi*}\zeta_2\theta_t)\sigma_0$;

so $\mathcal{L} \approx \mathcal{R}$.

QED

Next special schedules will be investigated which mimic serial operation. First, an obvious lemma indicating how a schedule can be made to force the whole of one command to be done before any of the others start. Note that in the case of non-terminating commands, the schedule in 1.4.2 and the Serial Schedules introduced later are not Fair, so the criterion of Fairness has to be weakened in this case to allow schedules which consist entirely of a single $\pi_0<\pi^*$ after a certain point. This has the effect that lemmas 1.1.4 and 1.2.3 are no longer true in general, but they are still true if modified to refer only to π_0, which turns out to be sufficient to prove the remaining results.

Lemma 1.4.2:

If $\pi<\pi^*$

and $\ell=\#(Expand(\theta^{\pi*}\pi)\sigma_0)$ is finite

and $\sigma_1=Outcome(\theta^{\pi*}\pi)\sigma_0$

and ζ satisfies

for $1\le\iota\le\ell$, $\zeta\Downarrow\iota=\pi$

then $Expand(Combine\theta^{\pi*}\zeta\theta_0)\sigma_0 =$

$$Expand(\theta^{\pi*}\pi)\sigma_0 \hat{} Expand(Combine\theta^{\pi*}[\theta_t/\pi](\zeta\dagger\ell)\theta_0)\sigma_1 .$$

Proof

by induction on ℓ.

If $\ell=0$, then $\theta^{\pi*}\pi=\theta_t$ and $\sigma_1=\sigma_0$, so result is trivial.

If $\ell>0$, then by 1.1.3, *Choose*$\theta^{\pi*}\zeta=\langle \pi,\zeta\dagger 1\rangle$;

let $\langle \chi,\theta'\rangle=\theta^{\pi*}\pi\sigma_0$;

so $\mathcal{L}=\langle \chi\rangle$ ^*Expand*(*Combine*$\theta'^{\pi*}(\zeta\dagger 1)(\chi\sigma_0)$)

where $\theta'^{\pi*}=\theta^{\pi*}[\theta'/\pi]$;

but $\#(Expand(\theta^{\pi*}\pi)\sigma_0)=\#(\langle \chi\rangle \hat{}\, Expand(\theta'^{\pi*})(\chi\sigma_0))$;

so $\#(Expand(\theta'^{\pi*}\pi)(\chi\sigma_0))=\ell-1$;

and $\sigma_1=Outcome(\theta'^{\pi*}\pi)(\chi\sigma_0)$;

and for $1\le\iota\le\ell-1$, $(\zeta\dagger 1)\downarrow\iota=\pi$;

so by inductive hypothesis,

$$\begin{aligned}\mathcal{L} &= \langle \chi\rangle \hat{}\, Expand(\theta'^{\pi*}\pi)(\chi\sigma_0) \\ &\qquad \hat{}\, Expand(Combine\theta'^{\pi*}[\theta_t/\pi](\zeta\dagger 1\dagger(\ell-1))\theta_0)\sigma_1 \\ &= Expand(\theta^{\pi*}\pi)\sigma_0 \hat{}\, Expand(Combine\theta^{\pi*}[\theta_t/\pi](\zeta\dagger\ell)\theta_0)\sigma_1 \ .\end{aligned}$$

The infinite case requires a similar modification to theorem 1.1

QED

This prompts the following definition:

Definition

For any σ_0 and Consistent $\theta^{\pi*}$

for $1\le\nu\le\#\pi^*$

let $\sigma_\nu=Outcome(\theta^{\pi*}\pi_\nu)\sigma_{\nu-1}$

and $\ell_\nu=\#(Expand(\theta^{\pi*}\pi_\nu)\sigma_{\nu-1})$

and for $0 \leq \nu \leq \#\pi^*$

$$\text{let } L_\nu = \sum_{\iota=1}^{\nu} \ell_\iota \ .$$

A *Serial Schedule* for θ^{π^*} and σ_0 is a schedule ζ_s, fair in π^*, which satisfies

for $1 \leq \nu \leq \#\pi^*$, for $L_{\nu-1} < \iota \leq L_\nu$, $\zeta_s \downarrow \iota = \pi_\nu$.

It is now possible to link the various results which may come from executing commands in parallel with the unique, well-defined result of executing them sequentially in the given order; more specifically to relate the following two very different definitions (in which γ denotes an element of $[C \to C]$):

$$Run_p \gamma^* \pi^* \theta =$$
$$Combine(\pi^*, \lambda \pi_\iota . \gamma_\iota \theta_t)\ (NewSchedule \pi^*)\ \theta$$

$$Run_s \gamma^* \pi^* \theta =$$
$$Fix(\lambda \psi . \lambda \iota \theta . \iota \leq \#\gamma^* \to \gamma_\iota(\psi(\iota+1)\theta), \theta) 1 \theta$$

The first of these uses all the mechanism set up in this chapter to describe the parallel execution of a list of commands, ($NewSchedule \pi^*$ produces an arbitrary schedule which is Fair in π^*,) and the second uses the continuation facility to describe their serial execution. (Note that the π^* argument in Run_s is redundant - it is included only for compatibilty with Run_p.)

The use of continuations in Run_s is indicative of the need to allow for jumps - one of the commands in a list destined for parallel execution cannot be permitted to jump to some point external to itself, and carry on, since in the serial case,

subsequent commands may never be executed, whereas in the parallel case, they will probably have already started, and there is no way of stopping them.

Thus a concept of "JumpFreeness" needs to be introduced. Unfortunately, although it is easy enough (at any rate in a well-structured language) to check syntactically whether a section of program contains a jump out of the section, it is generally impossible to tell whether or not a section will terminate with an error - some potential errors are checkable at compile time, but others invariably are not - eg the possibility of running out of storage space. So any useful definition of JumpFree needs to allow for the possibility of error. Without going into details of the structure of the state at this stage, it will be assumed that after any error, the state σ satisfies the predicate *ErrorSet*σ.

A slightly simplified JumpFree condition can now be defined as follows (a rather stronger one will be needed in the next chapter):

Definition

$\gamma \in [C \to C]$ is *X-JumpFree* if

$\forall \sigma_0$ either $\forall \theta$ *Expand*$(\gamma\theta)\sigma_0 =$ *Expand*$(\gamma\theta_t)\sigma_0 \hat{}$ *Expand*$\theta\sigma_1$

or $\forall \theta$ $\sigma_1 =$ *Outcome*$(\gamma\theta)\sigma_0$ and *ErrorSet*σ_1

where $\sigma_1 =$ *Outcome*$(\gamma\theta_t)\sigma_0$.

The equivalence defined in 1.3 needs similar extension to cope with error:

Definition

Two commands θ_1,θ_2 are *X-Equivalent* wrt σ_0 if

either $Expand\theta_1\sigma_0 \approx Expand\theta_2\sigma_0$

or $ErrorSet(Outcome\theta_1\sigma_0)$ and $ErrorSet(Outcome\theta_2\sigma_0)$.

Finally, the three conditions described above can be combined to give the desired equivalence.

Theorem 1.4:

If the components of $\gamma^*(\epsilon[C\rightarrow C]^*)$ are X-JumpFree

and $\#\gamma^*=\#\pi^*$,

and θ_0 and $\theta^{\pi^*}=\langle\pi^*,\lambda\pi_\iota.\gamma_\iota\theta_t\rangle$ are Consistent,

and $\langle\pi^*,\lambda\pi.Detail\theta^{\pi^*}\zeta\pi\sigma\rangle$ is Miscible for all ζ fair in π^*

then $Run_s\gamma^*\pi^*\theta_0$ and $Run_p\gamma^*\pi^*\theta_0$ are X-Equivalent wrt any σ_0 .

Proof

let $\zeta=NewSchedule\pi^*$

and ζ_s be a Serial Schedule for θ^{π^*} and σ_0 ;

then $Expand(Run_p\gamma^*\pi^*\theta_0)\sigma_0 = Expand(Combine\theta^{\pi^*}\zeta\theta_0)\sigma_0$

$\approx Expand(Combine\theta^{\pi^*}\zeta_s\theta_0)\sigma_0$

(by 1.4.1) ;

For $0\leq\nu\leq\#\pi^*$, let $\theta_\nu^{\pi^*}=\theta^{\pi^*}[\theta_t,\ldots,\theta_t/\pi_1,\ldots,\pi_\nu]$

and let $\psi=Fix(\lambda\psi.\lambda\iota\theta.\iota\leq\#\gamma^*\rightarrow\gamma_\iota(\psi(\iota+1)\theta),\theta)$

(so $Run_s\gamma^*\pi^*\theta_0=\psi1\theta_0$) ;

then it suffices to show (by induction on $\#\pi^*-\nu$) that

$Combine\theta_\nu^{\pi^*}(\zeta_s\dagger L_\nu)\theta_0$ and $\psi(\nu+1)\theta_0$

are X-Equivalent wrt σ_ν ;

let $\mathcal{L}$ denote $Expand(Combine\theta_\nu^{\pi^*}(\zeta_s\dagger L_\nu)\theta_0)\sigma_\nu$

and $\mathcal{R}$ $Expand(\psi(\nu+1)\theta_0)\sigma_\nu$;

If $\nu=\#\pi^*$ then $AreT\theta_\nu^{\pi^*}$;

so $\mathcal{L}=Expand\theta_0\sigma_\nu=\mathcal{R}$.

If $\nu<\#\pi^*$;

let $\pi=\pi_{\nu+1}$;

then $\ell_{\nu+1}=\#(Expand(\theta_\nu^{\pi*}\pi)\sigma_\nu)$;

and $\sigma_{\nu+1}=Outcome(\theta_\nu^{\pi*}\pi)\sigma_\nu$;

and for $1<\iota\leq\ell_{\nu+1}$, $(\zeta_s\dagger L_\nu)\downarrow\iota=\pi$;

so by 1.4.2,

$\mathcal{L}=Expand(\theta_\nu^{\pi*}\pi)\sigma_\nu\hat{}Expand(Combine\theta_{\nu+1}^{\pi*}(\zeta_s\dagger L_{\nu+1})\theta_0)\sigma_{\nu+1}$;

but by inductive hypothesis,

$Combine\theta_{\nu+1}^{\pi*}(\zeta_s\dagger L_{\nu+1})\theta_0$ and $\psi(\nu+2)\theta_0$

are X-Equivalent wrt $\sigma_{\nu+1}$;

hence if $\chi^*=Expand(\theta_\nu^{\pi*}\pi)\sigma_\nu\hat{}Expand(\psi(\nu+2)\theta_0)\sigma_{\nu+1}$

$=Expand(\gamma_{\nu+1}\theta_t)\sigma_\nu\hat{}Expand(\psi(\nu+2)\theta_0)\sigma_{\nu+1}$;

then either $\mathcal{L}\approx\chi^*$

or $ErrorSet(Apply\chi^*\sigma_\nu)$

and $ErrorSet(Apply\mathcal{L}\sigma_\nu)$;

but $\mathcal{R}=Expand(\gamma_{\nu+1}(\psi(\nu+2)\theta_0))\sigma_\nu$;

so by X-JumpFreeness of $\gamma_{\nu+1}$,

either $\mathcal{R}\approx\chi^*$

or $Apply\mathcal{R}\sigma_\nu=Apply\chi^*\sigma_\nu$ and $ErrorSet(Apply\chi^*\sigma_\nu)$;

hence either $\mathcal{L}\approx\chi^*\approx\mathcal{R}$

or $ErrorSet(Apply\mathcal{L}\sigma_\nu)$ and $ErrorSet(Apply\mathcal{R}\sigma_\nu)$;

hence $Combine\theta_\nu^{\pi*}(\zeta_s\dagger L_\nu)\theta_0$ and $\psi(\nu+1)\theta_0$

are X-Equivalent wrt σ_ν ;

in particular, for $\nu=0$,

$Combine\theta^{\pi*}\zeta_s\theta_0$ and $Run_s\gamma^*\pi^*\theta_0$ are X-Equivalent wrt σ_0 ;

ie $Run_p\gamma^*\pi^*\theta_0$ and $Run_s\gamma^*\pi^*\theta_0$ are X-Equivalent wrt σ_0 .

QED

CHAPTER 2: DEVELOPMENT OF SYNTACTIC CONDITIONS

"First it marked out a race-course, in a sort of circle... There was no 'One, two, three, and away!' but they began running when they liked, and left off when they liked, so that it was not easy to know when the race was over. However, when they had been running half an hour or so, the Dodo suddenly called out, 'The race is over!'"

Alice's Adventures in Wonderland: Chap 3.

2.0 More Notation and Domains

The last chapter having found semantic conditions on the use of parallelism, i.e. for the equivalence of Run_s and Run_p, this chapter will be concerned with starting to replace these conditions by syntactic ones, i.e. ones which can be applied by a compiler. The details of specific languages will not be considered until chapters 3, 4 and 5, but the techniques will be introduced, and some useful general results proved.

The notation used will again largely follow [23]: three letter names will be given to syntactic domains, e.g. Com Exp Dec, and elements thereof will be denoted by corresponding upper case greek letters - Θ E Δ. Again, subscripts and primes will be used to distinguish different elements of the same domain, and * will indicate (usually finite) lists. In semantic equations, syntactic entities will be enclosed in the special brackets ⟦ ⟧. New notation will be used for domains (e.g. states and environments) which are products - instead of identifying the components by use of the subscripting operator ↓, each component will have a unique name which will be specified in the definition of the domain structure thus:

$D = \text{namea}:D_\alpha \times \text{nameb}:D_\beta \times \ldots \times \text{namen}:D_\nu$.

These names will be used to select components by enclosing them in the "textual" brackets ⟦ ⟧, thus if $\delta \in D$ above, $\delta⟦\text{namei}⟧ \in D_\iota$. This means that domains can be modified by adding, removing or changing components without affecting references to other components, and similar domains with some of the same components can use the same selectors.

The use of [.../...] to indicate substitution will be extended to include these names, thus

if $\delta = \langle \delta_1, \delta_2, \ldots \delta_\nu \rangle \in D$

then $\delta[\delta'_\iota/\text{namei}] = \langle \delta_1, \ldots, \delta_{\iota-1}, \delta'_\iota, \delta_{\iota+1}, \ldots, \delta_\nu \rangle$

and if $D_\iota = A \to B$

then $\delta[\beta/\text{namei}/\alpha] = \langle \delta_1, \ldots, (\lambda\alpha'.\alpha=\alpha' \to \beta, \delta_\iota\alpha'), \ldots, \delta_\nu \rangle$

and the operator ^ will indicate that the new value is to be appended to a list, e.g.

if $D_\iota = A^*$

then $\delta[\alpha\hat{}/\text{namei}] = \langle \delta_1, \ldots, \langle \alpha \rangle \hat{}\delta_\iota, \ldots, \delta_\nu \rangle$

and if $D_\iota = A \to B^*$

then $\delta[\beta\hat{}/\text{namei}/\alpha] = \langle \ldots, (\lambda\alpha'.\alpha=\alpha' \to \langle \beta \rangle \hat{}\delta_\iota\alpha', \delta_\iota\alpha'), \ldots \rangle$.

As in the last chapter, α and β may be replaced by equal lengthed lists, (the elements of α^* being distinct,) to denote multiple substitution/appending.

Lastly note one anomalous use of *: if τ is a predicate on some domain W, then τ^* will denote the predicate on W^* given by:

$\tau^*\omega^* = \forall \iota \leq \#\omega^*, \tau\omega_\iota$.

Next, some new semantic domains. The domain L of locations is a flat lattice whose (proper) elements correspond roughly to memory locations in a real computer. (In fact, there is no reason why there should be a 1-1 correspondence between members

of L and actual memory addresses, though it is usually convenient to think of them in this way.) Then the model of the machine's store will be (or at least will contain as one component) a mapping from L to some set of possible storable values.

For the domain T of truth values, a flat lattice with *True* and *False* as its (only) proper elements is usually adequate; though for the few results which require fixed point inductions, a two point lattice is used, where *True* is identified with $\perp$ and *False* with $\top$. Note that $\top$ will be used both for elements of T and also for predicates, i.e. functions with range T.

The domain E represents all the possible values which can be produced as the result of evaluating an expression; it will normally be the sum of several basic domains, though its exact form varies from language to language. It will, however, be assumed that L is one of its summands. Lists of expressions E^* and lists of lists E^{**} will be needed, as will the domains $K=[E\to C]$ $K'=[E^*\to C]$ $K''=[E^{**}\to C]$ which will furnish "continuations" for the semantic equations dealing with expressions and declarations.

Lastly, the domain H of environments, whose main function is to give the current meaning or denotation of identifiers which have been defined in a program. It will also be convenient for an environment to have a component which is member of P, to say which process is executing the particular section of program under consideration. The denotation of an identifier may be either a data item or a continuation of some sort, depending on the context in which it is required. Only the latter will needed for the purposes of this chapter, so the structure of the environment domain will be:

H = proc:P × cont:[Ide→C] × ...

Before leaving the subject of domains, the domains P and S need closer examination. In the last chapter it was just assumed that P was a flat lattice (to ensure that equality is continuous). It is now necessary to introduce some structure on P other than its lattice structure. When a process is executing a task, it is liable to be called upon to spawn arbitrarily many "offspring" processes to perform subtasks. To control this, P is given a tree structure, whereby each element has a (countably infinite) number of (next generation) offspring. This makes P countably infinite in size, though of course only finitely many processes will ever be used, but this seems to be the simplest way of organising it - an alternative might be to have a pool of processes (kept as part of the state) and allow the tree structure to be built up as they are used. The notation $\pi_1 < \pi_2$ will be used to signify that π_1 is an ancestor (at any level) of π_2; and $\pi_1 \gtrless \pi_2$ to signify that π_1 and π_2 are incomparable with respect to this ordering. Hence in particular, if π_1 spawns π_2 as a sub-process, then $\pi_1 < \pi_2$. An important property of the processes used to execute a set of tasks in parallel is that they are incomparable according to this ordering, which, because of the tree structure, implies that descendants of any two are also incomparable. Thus it will be tacitly assumed hereafter that any list of processes π^* has the property that all its components are distinct and incomparable. In particular, the results of *NewProcs* will always satisfy this property.

One problem with a model of a parallel machine with a common store is that several parallel tasks may each claim new areas of store as working space, and exactly which locations each process

is assigned will depend on the relative timings of their activities. In practice, of course, this doesn't matter in the least since one storage location is as good as any other, but it makes for difficulties with equivalence proofs. One way round the difficulty is to demand weaker conditions than equality, which allow locations to be permuted (consistent, presumably, with some sort of map indicating which locations have been claimed by which process). A somewhat simpler, if apparently rather artificial, method which will be used here, is to "preassign" a number of locations to each process (probably 0 to most of them), and to allow a process to claim only locations which "belong" to it. (This does not debar other processes from using the location once claimed.) Thus a function $Owner \in [L \to P]$ will be assumed to exist, which defines this relationship; and the function *New* which claims a new location will take a process identifier as one argument (as will the predicate *StoreFull* which checks whether any locations are available) and return as result a location whose *Owner* is the given process (assuming one exists). i.e.

$\forall \pi\sigma$, unless $StoreFull\,\pi\sigma$

$\alpha = New\,\pi\sigma$ satisfies $Owner\,\alpha = \pi$

Use will also be made of the fact that $StoreFull\,\pi\sigma$ and $New\,\pi\sigma$ depend only on the values of $\sigma[\![\mathbf{area}]\!]\alpha$ for α such that $Owner\,\alpha = \pi$.

As for S, it will have as one component, as mentioned above, a mapping from L to storable values V. This domain V is similar to E in that it is the sum of other basic domains, though the summands of E and V need not be the same, since e.g.

not all expressions need have values that can legitimately be stored. Most languages also require the concept of whether a location is being used for anything, or whether it is free to be claimed for some new use. This appears as a mapping from L to T. The parallel processing features of the model require two more components - one which tells whether a process has been used for anything, and one where a process may leave its results for its parent process to look at later. This primitive system turns out to be all that is needed in the way of direct process communication. Lastly, any program may go wrong, so there must be some way of indicating that an error has been detected, and it is convenient that this should be another component of the state; furthermore, it is irrelevant for this investigation precisely what sort of error has occurred, so it will be sufficient to use an element of T which will just show whether or not an error has been found. The definition of *ErrorSet* should now be obvious. Errors have to be remembered in this way, rather than simply stopping the program immediately, so that any parallel processes can be stopped at some suitable point. Thus the structure of S is:

$$S = \mathbf{area}:[L \to T] \times \mathbf{store}:[L \to V] \times \mathbf{use}:[P \to T] \times \mathbf{res}:[P \to E] \times \mathbf{err}:T$$

Note the following special elements - the termination elements of K K′ K″ given by:

$$\kappa_t = \lambda\varepsilon.\theta_t$$
$$\kappa'_t = \lambda\varepsilon^{*}.\theta_t$$
$$\kappa''_t = \lambda\varepsilon^{**}.\theta_t$$

and the special error command:

$$Error = \lambda\sigma.\langle SetError, \theta_t \rangle$$

where $SetError = \lambda\sigma.\sigma[True/\mathrm{err}]$.

2.1 The Eval Series of Functions

In the last chapter, two forms (one for serial and one for parallel processing) of a function *Run* were given, which acted on elements of [C→C] which would be produced from sections of a program. Now that environments have been introduced, this can be taken one stage nearer the original program by defining functions *Eval* which take arguments in [H→C→C]. Not only is this form more convenient for use in the semantic equations which typically define functions of the form [Com→H→C→C], but it can also be extended to include arguments from [H→K→C] and [H→K′→C]. Furthermore, since most versions of *Eval* will need to manipulate processes explicitly, the environment (including as it does the current process) also needs to be explicit. *Eval* will therefore be a polymorphic function with functionalities

$$[[H\to C\to C]^*\to H\to C\to C]\ [[H\to K\to C]^*\to H\to K'\to C]\ [[H\to K'\to C]^*\to H\to K''\to C].$$

The definitions of the first two of these will be given below, the third will be identical to the second except for obvious minor modifications to the variables. Most of the results of this chapter will be stated and proved for the second case only: the third requiring only trivial modifications, and the first simply requiring the omission of certain parts referring to elements of E.

All versions of *Eval* can be defined in terms of one of other

of the versions of *Run*, though for the first two, *Run* needs to be made polymorphic also, viz:

$$Run_S\gamma^*\pi^*\kappa' =$$
$$Fix(\lambda\psi.\lambda\iota\kappa'.\iota\leq\#\gamma^* \rightarrow \gamma_\iota(\lambda\varepsilon.\psi(\iota+1)(\lambda\varepsilon^*.\kappa'(\langle\varepsilon\rangle\hat{}\varepsilon^*))),\ \kappa'\langle\rangle)1\kappa'$$

and similarly for $Run_S\gamma^*\pi^*\kappa''$.

The simplest form of *Eval* just applies Run_S in the obvious way:

$$Eval_1\rho^*\eta\theta = Run_S(\mathop{\#}_{\iota=1}^{\#\rho^*}\rho_\iota\eta)\langle\rangle\theta$$

$$Eval_1\rho^*\eta\kappa' = Run_S(\mathop{\#}_{\iota=1}^{\#\rho^*}\rho_\iota\eta)\langle\rangle\kappa'$$

This corresponds to the usual method of serial processing; it is not, however, very helpful for comparison with parallel operation, in which sub-processes must be initiated and communicate their results. So the next stage is to introduce into *Eval* operations for manipulating elements of P: there are three operations required - claiming of new processes (the function *NewProcs*$\pi\nu\sigma$ returns a list of ν distinct processes π^* which are not marked as used in σ, and which are all offspring of π); communicating of results by means of the special component of the state; and fetching a list of results out of the state again. It will also be necessary, as indicated above, for a parent process to check whether any of its offspring has found an error, since in the parallel situation a process may discover an error and stop, but it cannot affect its siblings' operation. (Aborting of one process by another requires a very complicated mechanism both to model

theoretically and also to implement, and doesn't seem necessary, so it is not considered.)

Hence four types of primitive operation are needed: the first for putting a value calculated by a process into the space reserved for it in the ⟦res⟧ component of the state; the other three for ensuring Consistency when new processes are claimed, when results are fetched from the state, and when the parent process checks to see if an error has been detected.

Definitions

$Signal\pi\varepsilon\sigma = \sigma[\varepsilon/\mathbf{res}/\pi]$

$CheckProcs\pi\pi^*\sigma = (NewProcs\pi(\#\pi^*)\sigma{=}\pi^* \rightarrow \sigma[True^*/\mathbf{use}/\pi^*],\ \underline{?})$

$CheckResults\pi^*\varepsilon^*\sigma = (Results\pi^*\sigma{=}\varepsilon^* \rightarrow \sigma,\ \underline{?})$

$CheckError\tau\sigma = (\sigma[\![\mathbf{err}]\!]{=}\tau \rightarrow \sigma,\ \underline{?})$

$$Results\pi^*\sigma = \mathop{\Large *}_{\iota=1}^{\#\pi^*} (\sigma[\![\mathbf{res}]\!]\pi_\iota)$$

$Result\pi\sigma = \sigma[\![\mathbf{res}]\!]\pi$

and $NewProcs\pi\nu\sigma$ gives some $\pi^* \in P^*$ satisfying

$\#\pi^* = \nu$;

the components of π^* are distinct and incomparable (by the tree ordering on P);

and $\forall \pi_\iota \lhd \pi^*,\ \pi < \pi_\iota$;

and $True^*$ denotes a list of $\#\pi^*$ copies of *True*.

$$Eval_2\rho^*\eta\theta =$$

$$ClaimProcs(\eta[\![\,\mathsf{proc}\,]\!])(\#\rho^*)\{\lambda\pi^*.Run_s(\mathop{\ast}_{\iota=1}^{\#\rho^*}\rho_\iota\eta[\pi_\iota/\mathsf{proc}])\pi^*\theta\}$$

$$Eval_2\rho^*\eta\kappa' =$$

$$ClaimProcs(\eta[\![\,\mathsf{proc}\,]\!])(\#\rho^*)$$

$$\{\lambda\pi^*.Run_s(\mathop{\ast}_{\iota=1}^{\#\rho^*}\lambda\kappa.\rho_\iota\eta[\pi_\iota/\mathsf{proc}](\lambda\varepsilon\sigma.\langle Signal\pi_\iota\varepsilon,\kappa\varepsilon\rangle))\pi^*$$

$$\{\lambda\varepsilon^*.UnlessError(\lambda\sigma.\langle CheckResults\pi^*\varepsilon^*,\kappa'\varepsilon^*\rangle)\}\}$$

where $ClaimProcs\pi\nu\gamma\sigma = (\lambda\pi^*.\langle CheckProcs\pi\pi^*,\gamma\pi^*\rangle)$

$$(NewProcs\pi\nu\sigma)$$

$$UnlessError\theta\sigma = (\lambda\tau.\langle CheckError\tau,(\tau\rightarrow\theta_t,\theta)\rangle)$$

$$(\sigma[\![\,\mathsf{err}\,]\!])\ .$$

Note that, at this stage, apart from putting the process identifier into the environment and the lack of Consistency introduced by $CheckResults\pi^*\varepsilon^*$, no use has been made of the processes; primitive operations have been added which manipulate the "process" part of the state, but these have no significant effect on the normal running of a program.

The next stage, however, is to make use of these "process" components to remove the need for the "expression" and "expression list" versions of *Run*, viz.

$$Eval_3\rho^*\eta\theta\sigma = Eval_2\rho^*\eta\theta\sigma$$

$$Eval_3\rho^*\eta\kappa'\sigma =$$

$$ClaimProcs(\eta[\![\mathbf{proc}]\!])(\#\rho^*)$$

$$\lambda\pi^*.Run_s(\overset{\#\rho^*}{\underset{\iota=1}{\ast}}\lambda\theta.\rho_\iota\eta[\pi_\iota/\mathbf{proc}](\lambda\varepsilon\sigma.\langle Signal\pi_\iota\varepsilon,\theta\rangle))\pi^*$$

$$\{UnlessError(\lambda\sigma.(\lambda\varepsilon^*.\langle CheckResults\pi^*\varepsilon^*,\kappa'\varepsilon^*\rangle)$$

$$(Results\pi^*\sigma))\}\}$$

This change restores Consistency (see 2.2), and as will be seen later, $Eval_2$ and $Eval_3$ are to all intents and purposes equivalent.

Finally, the parallel version $Eval_4$ is obtained simply by replacing Run_s by Run_p in the definition of $Eval_3$.

As indicated by theorem 1.4, conditions have to be imposed on the use of Run_p, and therefore similarly on the parallel version of *Eval*. As will be demonstrated in the next chapters, sufficient conditions can be expressed purely in terms of syntactic objects. Then two new functions can be defined which take a list of syntactic elements as first parameter, one executes them sequentially, the other uses the syntactic conditions to decide whether to execute them sequentially or in parallel. These functions will inevitably be specific to one syntactic domain and one semantic function; at this stage they will be described with reference to a general semantic function X acting on elements Ξ of a general syntactic domain Xxx.

For the present, these functions will be defined in terms of (unspecified) predicates on the syntactic domains, known as P-Tests, which will ensure that the hypotheses of theorem 1.4 hold. Chapters 3, 4 and 5 will then develop non-trivial P-Tests (the constant *False* function is always a P-Test, but not

a very useful one!) for the three example languages. Before defining P-Tests, generalised versions of *Expand*, *Outcome* and *Apply* are required, which allow for other operations to be happening simultaneously with the command under consideration, by taking an explicit list of (not necessarily primitive) operations representing the action of all other processes; and also an extension of the concept of JumpFreeness, which not only uses these new functions, but also allows for expression and expression list continuations.

Definitions

$$Develop\theta\chi^*\sigma =$$
$$(\theta=\theta_t \rightarrow \langle\rangle\ ,\ (\lambda\langle\chi,\theta'\rangle.\langle\chi\rangle\ \hat{}\ Develop\theta'(\chi^*\dagger 1)(\chi_1\circ\chi\sigma))(\theta\sigma))$$

$$Consequence\theta\chi^*\sigma =$$
$$(\theta=\theta_t \rightarrow \sigma,\ (\lambda\langle\chi,\theta'\rangle.Consequence\theta'(\chi^*\dagger 1)(\chi_1\circ\chi\sigma))(\theta\sigma))$$

$$Digest\chi'^*\chi^*\sigma =$$
$$(\#\chi'^*=0 \rightarrow \sigma,\ Digest(\chi'^*\dagger 1)(\chi^*\dagger 1)(\chi_1\circ\chi_1'\sigma))$$

Analogous results to those of section 1.1 are easy to prove, eg:

$$Consequence\theta\chi^*\sigma = Digest(Develop\theta\chi^*\sigma)\chi^*\sigma$$

$$Digest(\chi_1^*\hat{}\chi_2^*)\chi^* = Digest\chi_2^*(\chi^*\dagger\nu)\circ Digest\chi_1^*\chi^*$$
where $\nu=\#\chi_1^*$

Definition

$\gamma\epsilon[K\to K]$ is *JumpFree* if

$\forall\chi^*,\sigma_0,\varepsilon,$

either $\exists\varepsilon'$, $\forall\kappa$, $Develop(\gamma\kappa\varepsilon)\chi^*\sigma_0 =$

$Develop(\gamma\kappa_t\varepsilon)\chi^*\sigma_0\hat{}Develop(\kappa\varepsilon')(\chi^*\dagger\nu)\sigma_1$

or $Develop(\gamma\kappa\varepsilon)\chi^*\sigma_0$ is independent of κ, and $ErrorSet\sigma_1$

where $\sigma_1=Consequence(\gamma\kappa_t\varepsilon)\chi^*\sigma_0$

and $\nu=\#Develop(\gamma\kappa_t\varepsilon)\chi^*\sigma_0$

and similarly for $\gamma\epsilon[C\to C]$ $[C\to K]$ $[K\to C]$ etc.

Note that *Expand*, *Outcome* and *Apply* are special cases of *Develop*, *Consequence* and *Digest* respectively, and that JumpFree implies X-JumpFree.

It will be shown later (2.3.1) that *Detail* can also be viewed as a special case of *Develop*; hence the hypotheses imposed on Expansions and Details in theorem 1.4 follow from similar conditions on Developments.

A provisional definition can now be given of a P-Test, (though it will be refined slightly in section 2.3,) and the result of 1.4 rephrased in terms of it.

Definition

A *P-Test* for the functions $X_\iota\epsilon[X\kappa x\to H\to C\to C]$ $(1\le\iota\le\nu)$ wrt τ' is a function $\tau\epsilon[X\kappa x^*\to H\to T]$ which satisfies:

$\forall\Xi^*\epsilon X\kappa x^*,\pi^*,\zeta,\eta,\sigma$ s.t.

$\#\Xi^*=\#\pi^*$, $\tau'\eta$ and each $\pi_\iota>\eta[\![\,proc\,]\!]$

$\tau[\![\,\Xi^*\,]\!]\eta \Rightarrow$ for $1\le\iota\le\nu$ if $\eta_\iota=\eta[\pi_\iota/proc]$

$X_\iota[\![\,\Xi_\iota\,]\!]\eta_\iota$ is JumpFree and preserves Consistency;

and either $\chi^{*\pi*}$ is Miscible

or $ErrorSet(Apply(Mix\chi^{*\pi*}\zeta)\sigma)$

where $\chi^{*\pi*}=\langle \pi^*, \lambda\pi.Detail\theta^{\pi*}\zeta\pi\sigma\rangle$

and $\theta^{\pi*}=\langle \pi^*, \lambda\pi_\iota.X_\iota[\![\Xi_\iota]\!]\eta_\iota\theta_t\rangle$;

and similarly for $X_\iota\epsilon[Xxx\rightarrow H\rightarrow K\rightarrow C]$

with $\theta^{\pi*}=\langle \pi^*, \lambda\pi_\iota.X_\iota[\![\Xi_\iota]\!]\eta_\iota(\lambda\varepsilon\sigma.\langle Signal\pi_\iota\varepsilon,\theta_t\rangle)\rangle$.

Then for any set of functions

$X_\iota\epsilon[Xxx\rightarrow H\rightarrow C\rightarrow C] + [Xxx\rightarrow H\rightarrow K\rightarrow C] + [Xxx\rightarrow H\rightarrow K'\rightarrow C]$:

$$XEval_s[\![\Xi^*]\!]\eta = Eval_3(\mathop{\divideontimes}_{\iota=1}^{\#\Xi^*} X_\iota[\![\Xi_\iota]\!])\eta$$

$$XEval_p[\![\Xi^*]\!]\eta =$$

$$(\tau[\![\Xi^*]\!]\eta \rightarrow Eval_4(\mathop{\divideontimes}_{\iota=1}^{\#\Xi^*} X_\iota[\![\Xi_\iota]\!])\eta,\ Eval_3(\mathop{\divideontimes}_{\iota=1}^{\#\Xi^*} X_\iota[\![\Xi_\iota]\!])\eta$$

Theorem 2.1:

If $XEval_p$ is defined in terms of a P-Test τ wrt τ'

then for any Consistent $\psi\epsilon[C+K'+K'']$ and η s.t. $\tau'\eta$,

$(XEval_s[\![\Xi^*]\!]\eta\psi)$ and $(XEval_p[\![\Xi^*]\!]\eta\psi)$ are X-Equivalent.

Proof

For any σ_0,

let $\mathcal{L}=Expand(XEval_s[\![\Xi^*]\!]\eta\psi)\sigma_0$

and $\mathcal{R}=Expand(XEval_p[\![\Xi^*]\!]\eta\psi)\sigma_0$.

If $\tau[\![\Xi^*]\!]\eta\neq True$

then $\mathcal{R} = Expand(Eval_3(\mathop{\divideontimes}_{\iota=1}^{\#\Xi^*} X_\iota[\![\Xi_\iota]\!])\eta\psi)\sigma_0$

$= \mathcal{L}$.

If $\tau[\![\Xi^*]\!]\eta = \mathit{True}$;

let $\pi = \eta[\![\mathbf{proc}]\!]$ and $\nu = \#\Xi^*$

and $\pi^* = \mathit{NewProcs}\pi\nu\sigma_0$;

for $1 \le \iota \le \nu$

let $\rho_\iota = X_\iota[\![\Xi_\iota]\!]$

and $\eta_\iota = \eta[\pi_\iota/\mathbf{proc}]$

and $\gamma_\iota = \rho_\iota\eta_\iota$.

If $\psi \in C$

then by definition of P-Test,

each γ_ι is JumpFree;

$\theta^{\pi*} = \langle \pi^*, \lambda\pi_\iota.\gamma_\iota\theta_t \rangle$ is Consistent;

and for any ζ fair in π^*, and σ'

$\langle \pi^*, \lambda\pi.\mathit{Detail}\theta^{\pi*}\zeta\pi\sigma' \rangle$ is Miscible;

hence either both $\mathit{ErrorSet}(\mathit{Apply}\pounds\sigma_0)$

and $\mathit{ErrorSet}(\mathit{Apply}\mathcal{R}\sigma_0)$

or $\pounds$ - $\mathit{Expand}(\mathit{Eval}_3\rho^*\eta\psi)\sigma_0$

$= \langle \mathit{CheckProcs}\pi\pi^* \rangle \hat{}\, \mathit{Expand}(\mathit{Run}_s\gamma^*\pi^*\psi)\sigma_1$

where $\sigma_1 = \mathit{CheckProcs}\pi\pi^*\sigma_0$

$\approx \langle \mathit{CheckProcs}\pi\pi^* \rangle \hat{}\, \mathit{Expand}(\mathit{Run}_p\gamma^*\pi^*\psi)\sigma_1$

$\approx \mathit{Expand}(\mathit{Eval}_4\rho^*\eta\psi)\sigma_0$

$\approx \mathcal{R}$;

If however $\psi \in K'$ (the proof for $\psi \in K''$ is similar);

let $\gamma'_\iota = \lambda\theta.\gamma_\iota(\lambda\varepsilon\sigma.\langle \mathit{Signal}\pi_\iota\varepsilon, \theta \rangle)$;

again, each γ_ι is JumpFree,

so either $\mathit{Expand}(\gamma_\iota\kappa)\sigma_0$ is independent of κ

and $\mathit{ErrorSet}(\mathit{Outcome}(\gamma_\iota\kappa_t)\sigma_0)$;

in which case, $\mathit{Expand}(\gamma'_\iota\theta)\sigma_0$ is independent of θ ;

or let ε_ι be s.t.

$$\forall\kappa,\ \mathit{Expand}(\gamma_\iota\kappa)\sigma_0 = \mathit{Expand}(\gamma_\iota\kappa_t)\sigma_0\hat{}\mathit{Expand}(\kappa\varepsilon_\iota)\sigma_1$$

where $\sigma_1 = \mathit{Outcome}(\gamma_\iota\kappa_t)\sigma_0$;

then for any θ,

$$\begin{aligned}\mathit{Expand}(\gamma'_\iota\theta)\sigma_0 &= \mathit{Expand}(\gamma_\iota(\lambda\varepsilon\sigma.\langle \mathit{Signal}\pi_\iota\varepsilon,\theta\rangle))\sigma_0\\ &= \mathit{Expand}(\gamma_\iota\kappa_t)\sigma_0\hat{}\\ &\qquad \mathit{Expand}(\lambda\sigma.\langle \mathit{Signal}\pi_\iota\varepsilon_\iota,\theta\rangle)\sigma_1\\ &= \mathit{Expand}(\gamma_\iota\kappa_t)\sigma_0\hat{}\langle \mathit{Signal}\pi_\iota\varepsilon_\iota\rangle\hat{}\mathit{Expand}\theta\sigma_2\\ &\qquad \text{where } \sigma_2 = \mathit{Signal}\pi_\iota\varepsilon_\iota\sigma_1\\ &\qquad\qquad = \mathit{Outcome}(\gamma'_\iota\theta_t)\sigma_0\\ &= \mathit{Expand}(\gamma'_\iota\theta_t)\sigma_0\hat{}\mathit{Expand}\theta\sigma_2\ ;\end{aligned}$$

hence in either case γ'_ι is JumpFree;

also $\lambda\varepsilon\sigma.\langle \mathit{Signal}\pi_\iota\varepsilon,\theta_t\rangle$ is certainly Consistent;

so $\theta^{\pi*}=\langle \pi^*,\lambda\pi_\iota.\gamma'_\iota\theta_t\rangle$ is too;

also by definition of P-Test,

$\chi^{*\pi*}=\langle \pi^*,\lambda\pi.\mathit{Detail}\theta^{\pi*}\zeta\pi\sigma_0\rangle$ is Miscible

where $\theta^{\pi*}=\langle \pi^*,\lambda\pi_\iota.\gamma'_\iota\theta_t\rangle$;

hence the γ'_ι satisfy the hypotheses of 1.4,

and the result follows as in the case $\psi\epsilon C$.

QED

2.2 Preservation of Consistency

The next few sections will introduce techniques for ensuring that the conditions of theorem 1.4 are satisfied, usually by introducing syntactic conditions; and at the same time various results will be proved about the various versions of *Eval*, to the effect that they preserve certain desirable properties.

The first and easiest condition is that of Consistency; the semantic functions are all designed to produce universally Consistent results, and a simple application of structural induction on the main syntactic categories, together with the following theorem, suffices to prove Consistency with no extra conditions needing to be imposed.

First, note the following obvious extensions of Consistency to expression continuations and environments:

κ (or κ') is ν-Consistent if

$\kappa\varepsilon$ ($\kappa'\varepsilon^*$) is ν-Consistent for all ε (ε^*);

η is ν-Consistent if

$\eta[\![\mathbf{cont}]\!][\![I]\!]$ is ν-Consistent for all I.

Lemma 2.2.1:

If each γ_ι preserves ν-Consistency and θ is ν-Consistent then $Run_s\gamma^*\pi^*\theta$ is ν-Consistent.

Proof

let $\psi = Fix(\lambda\psi.\lambda\iota\theta.\iota\leq\#\gamma^* \rightarrow \gamma_\iota(\psi(\iota+1)\theta), \theta)$;

then it will be shown, by induction on $(\#\gamma^*-\iota)$, that

for each ι, $\psi\iota\theta$ is ν-Consistent.

if $\iota>\#\gamma^*$

then $\psi\iota\theta=\theta$ which is ν-Consistent.

if $\iota\leq\#\gamma^*$

then by induction, $\psi(\iota+1)\theta$ is ν-Consistent;

so $\psi\iota\theta = \gamma_\iota(\psi(\iota+1)\theta)$ is ν-Consistent;

hence $\psi\iota\theta$ is ν-Consistent for all $\iota \geq 1$;

in particular, $Run_s\gamma^*\pi^*\theta = \psi 1\theta$ is ν-Consistent.

QED

Lemma 2.2.2:

If each γ_ι preserves ν-Consistency and θ is ν-Consistent

then $Run_p\gamma^*\pi^*\theta$ is ν-Consistent.

Proof

let $\theta^{\pi*} = \langle \pi^*, \lambda\pi_\iota.\gamma_\iota\theta_t \rangle$ which is ν-Consistent;

let $\zeta = NewSchedule\pi^*$

and let $\theta' = Run_p\gamma^*\pi^*\theta = Combine\theta^{\pi*}\zeta\theta$;

then (by induction on ν) θ' is ν-Consistent, since:

if $AreT\theta^{\pi*}$, then $\theta' = \theta$ which is ν-Consistent;

if $\nu = 0$ then the result is immediate from the definition;

otherwise take any σ_1, σ_2;

let $\langle \pi, \zeta' \rangle = Choose\theta^{\pi*}\zeta$

and $\langle \chi_\iota, \theta_\iota \rangle = \theta^{\pi*}\pi\sigma_\iota$ (for $\iota = 1, 2$);

so $\theta'\sigma_\iota = \langle \chi_\iota, Combine\theta^{\pi*}[\theta_\iota/\pi]\zeta'\theta \rangle$;

then a) by ν-Consistency of $\theta^{\pi*}\pi$, θ_1 is $(\nu-1)$-Consistent;

so $\theta^{\pi*}[\theta_1/\pi]$ is $(\nu-1)$-Consistent;

so by inductive hypothesis

$Combine\theta^{\pi*}[\theta_1/\pi]\zeta'\theta$ is $(\nu-1)$-Consistent;

ie $\theta'\sigma_1\downarrow 2$ is $(\nu-1)$-Consistent;

b) $(\theta'\sigma_1\downarrow 1)\sigma_1 = \chi_1\sigma_1 \neq^?$ from ν-Consistency of $\theta^{\pi*}\pi$;

c) from ν-Consistency of $\theta^{\pi*}\pi$,

either $\chi_1 = \chi_1$ and $\theta_1 = \theta_2$;

in which case $\theta'\sigma_1 = \theta'\sigma_2$;

or $\chi_2\sigma_1 =\stackrel{?}{=}$;

in which case $(\theta'\sigma_2\downarrow 1)\sigma_1 = \chi_2\sigma_1 =\stackrel{?}{=}$;

Hence θ' is ν-Consistent.

QED

Theorem 2.2:

$Eval_3$ and $Eval_4$ preserve Consistency.

ie if ρ^* satisfies:

for $1 \le \iota \le \#\rho^*$ and all ν-Consistent η and θ (or κ);

$\rho_\iota \eta\theta$ (or $\rho_\iota \eta\kappa$) is ν-Consistent;

and if η and θ (or κ') are ν-Consistent

then $Eval\rho^*\eta\theta$ (or $Eval\rho^*\eta\kappa'$) is ν-Consistent.

Proof

consists of the simple verification that:

1) if κ' is ν-Consistent, then so is

$UnlessError(\lambda\sigma.\langle CheckResults\pi^*\varepsilon^*, \kappa'\varepsilon^*\rangle$

where $\varepsilon^* = Results\pi^*\sigma$;

2) if θ' is ν-Consistent, then so is $(\lambda\varepsilon\sigma.\langle Signal\pi_\iota\varepsilon, \theta'\rangle)$;

and therefore so is $\rho_\iota\eta[\pi_\iota/\mathbf{proc}](\lambda\varepsilon\sigma.\langle Signal\pi_\iota\varepsilon, \theta'\rangle)$;

3) *ClaimProcs* preserves Consistency;

and application of the appropriate lemma (2.2.1 or 2.2.2).

QED

2.3 Termination

The next stage is to examine more carefully in what possible ways a piece of program may finish its execution. It will be shown that there are three basic ways - "normal" termination, so that the rest of the program (the continuation) is then executed as usual; a jump to some point external to the section under consideration, and it is assumed that this point can be designated in some way by an identifier; or detection of an error, in which case it doesn't really matter what happens next, though pragmatically it is desirable that execution should cease as soon as possible so that the error can be investigated. Which of these three possibilities occurs in any given situation, and the set of possibilities which may arise from a given program section, can be registered as follows, using two special new symbols ✓ and ?

Definition

$I \in [\mathbf{Ide} + \{✓\} + \{?\}]$ is a *Terminator* for $\rho, \eta', \chi^*, \sigma_0$

where $\rho \in [H \to K \to C]$ and $\eta'[\![\mathbf{cont}]\!] = \lambda I.\theta_t$;

if $\forall \eta$ satisfying $\eta' = \eta[\lambda I.\theta_t / \mathbf{cont}]$,

$\forall \kappa$, $Develop(\rho\eta\kappa)\chi^*\sigma_0$ -

$Develop(\rho\eta'\kappa_t)\chi^*\sigma_0 \hat{} Develop\theta'(\chi^* \dagger \nu)\sigma_1$

where $\nu = \#Develop(\rho\eta'\kappa_t)\chi^*\sigma_0$

and $\sigma_1 = Consequence(\rho\eta'\kappa_t)\chi^*\sigma_0$;

and if $I = ✓$ then $\theta' = \kappa\varepsilon$ for some ε independent of η, κ;

if $I \in \mathbf{Ide}$ then $\theta' = \eta[\![\mathbf{cont}]\!][\![I]\!]$;

if $I = ?$ then $\theta' = \theta_t$ and $ErrorSet\sigma_1$;

and similarly for $\rho \in [H \to C \to C]$ and $\rho \in [H \to K' \to C]$.

Definition

$I^* \in Ide^*$ is a *T-List* for $\rho \in [H \to K \to C]$ if

$\forall \eta', \chi^*, \sigma_0$ satisfying $\eta'[\![\mathbf{cont}]\!] = \lambda I.\theta_t$,

some $I \lessdot I^*$ or $\surd$ or ? is a Terminator for $\rho, \eta', \chi^*, \sigma_0$.

Clearly, if ρ has an empty T-List, then $\rho\eta$ is JumpFree for all η. So the definition of a P-Test can now be modified to replace the JumpFree condition by the slightly stronger one of an empty T-List; since this condition is stronger, all the previous results still, of course, hold, but the new version allows lemma 2.3.2 to take the fairly simple form it does.

Definition

A *P-Test* for the functions $X_\iota \in [X \times X \to H \to C \to C]$ $(1 \le \iota \le \nu)$ wrt τ' is a function $\tau \in [X \times X^* \to H \to T]$ which satisfies

$\forall \Xi^* \in X \times X^*, \pi^*, \zeta, \eta$ s.t. $\#\Xi^* = \#\pi^*$, $\tau'\eta$ and each $\pi_\iota > \eta[\![\mathbf{proc}]\!]$

$\tau[\![\Xi^*]\!]\eta \Rightarrow$ for $1 \le \iota \le \nu$ if $\eta_\iota = \eta[\pi_\iota / \mathbf{proc}]$,

$\langle\rangle$ is a T-List for $X_\iota[\![\Xi_\iota]\!]$

and $X_\iota[\![\Xi_\iota]\!]\eta_\iota$ preserves Consistency;

and either $\chi^{*\pi^*}$ is Miscible

or $ErrorSet(Apply(Mix\chi^{*\pi^*}\zeta)\sigma)$

where $\chi^{*\pi^*} = \langle \pi^*, \lambda\pi.Detail\theta^{\pi^*}\zeta\pi\sigma \rangle$

and $\theta^{\pi^*} = \langle \pi^*, \lambda\pi_\iota.X_\iota[\![\Xi_\iota]\!]\eta_\iota\theta_t \rangle$;

and similarly for $X_\iota \in [X \times X \to H \to K \to C]$

with $\theta^{\pi^*} = \langle \pi^*, \lambda\pi_\iota.X_\iota[\![\Xi_\iota]\!]\eta_\iota(\lambda\varepsilon\sigma.\langle Signal\pi_\iota\varepsilon, \theta_t \rangle) \rangle$.

The rest of this section is devoted to showing that T-Lists are preserved by *Run* and *Eval*.

Lemma 2.3.1:

If I^* is a T-List for each ρ_ι

then it is a T-List for $\rho=\lambda\eta.Run_s(\overset{\#\rho^*}{\underset{\iota=1}{\divideontimes}}\rho_\iota\eta)\pi^*$.

Proof

let $\psi_\eta=Fix(\lambda\psi.\lambda\iota\theta.\iota\leq\#\rho^*\rightarrow\rho_\iota\eta(\psi(\iota+1)\theta),\theta)$;

then $\rho=\lambda\eta.\psi_\eta 1$;

so it suffices to show (by induction on $\#\rho^*-\iota$) that

I^* is a T-List for $\rho'_\iota=\lambda\eta.\psi_\eta\iota$.

If $\iota>\#\rho^*$ then $\rho'_\iota=\lambda\eta\theta.\theta$;

for which ✓ is trivially a Terminator for any η',χ^*,σ_0 .

If $\iota\leq\#\rho^*$ then $\rho'_\iota=\lambda\eta\theta.\rho_\iota\eta(\psi_\eta(\iota+1)\theta)$;

Take any η',χ^*,σ_0 s.t. $\eta'[\![\mathbf{cont}]\!]=\lambda I.\theta_t$;

and in what follows η will always satisfy

$\eta'=\eta[\lambda I.\theta_t/\mathbf{cont}]$;

let I_1 be a Terminator for $\rho_\iota,\eta',\chi^*,\sigma_0$;

then $\forall\eta,\theta$, $Develop(\rho'_\iota\eta\theta)\chi^*\sigma_0$

$= Develop(\rho_\iota\eta(\psi_\eta(\iota+1)\theta))\chi^*\sigma_0$

$= Develop(\rho_\iota\eta'\theta_t)\chi^*\sigma_0\hat{}Develop\theta'(\chi^*\dagger\nu_1)\sigma_1$;

If $I_1\in\mathbf{Ide}$ then $\theta'=\eta[\![\mathbf{cont}]\!][\![I_1]\!]$;

so $Develop(\rho'_\iota\eta'\theta_t)\chi^*\sigma_0=Develop(\rho_\iota\eta'\theta_t)\chi^*\sigma_0$;

and $Develop(\rho'_\iota\eta\theta)\chi^*\sigma_0 = Develop(\rho'_\iota\eta'\theta_t)\chi^*\sigma_0$

$\hat{}Develop(\eta[\![\mathbf{cont}]\!][\![I_1]\!])(\chi^*\downarrow\nu_1)\sigma_1$;

so $I_1\lhd I^*$ is a Terminator for $\rho'_\iota,\eta',\chi^*,\sigma_0$.

if $I_1=?$ then $\theta'=\theta_t$ and $ErrorSet\sigma_1$;

so $Develop(\rho'_\iota\eta'\theta_t)\chi^*\sigma_0=Develop(\rho_\iota\eta'\theta_t)\chi^*\sigma_0$;

so $Develop(\rho_\iota'\eta\theta)\chi^*\sigma_0 = Develop(\rho_\iota'\eta'\theta_t)\chi^*\sigma_0$;

and $ErrorSet(Consequence(\rho_\iota'\eta'\theta_t)\chi^*\sigma_0)$;

so ? is a Terminator for $\rho_\iota', \eta', \chi^*, \sigma_0$;

if $I_1 = \surd$ then $\theta' = \psi_\eta(\iota+1)\theta$;

but by inductive hypothesis,

I^* is a T-List for $\rho_{\iota+1}'$;

so let I_2 be a Terminator for $\rho_{\iota+1}', \eta', \chi'^*, \sigma_1$

where $\chi'^* = \chi^* \dagger \nu_1$;

then $Develop(\rho_\iota'\eta\theta)\chi^*\sigma_0$

$= Develop(\rho_\iota\eta'\theta_t)\chi^*\sigma_0 \hat{} Develop(\psi_\eta(\iota+1)\theta)\chi'^*\sigma_1$

$= Develop(\rho_\iota\eta'\theta_t)\chi^*\sigma_0 \hat{} Develop(\rho_{\iota+1}'\eta\theta)\chi'^*\sigma_1$

$= Develop(\rho_\iota\eta'\theta_t)\chi^*\sigma_0 \hat{} Develop(\rho_{\iota+1}'\eta'\theta_t)\chi'^*\sigma_1$

$\hat{} Develop\theta''(\chi'^* \dagger \nu_2)\sigma_2$

where if $I_2 \in$ Ide then $\theta'' = \eta[\![\text{cont}]\!][\![I_2]\!]$;

if $I_2 = \surd$ then $\theta'' = \theta$;

if $I_2 = ?$ then $\theta'' = \theta_t$ and $ErrorSet\sigma_2$;

so in particular,

$Develop(\rho_\iota'\eta'\theta_t)\chi^*\sigma_0$

$= Develop(\rho_\iota\eta'\theta_t)\chi^*\sigma_0 \hat{} Develop(\rho_{\iota+1}'\eta'\theta_t)\chi'^*\sigma_1$;

so $Develop(\rho_\iota'\eta\theta)\chi^*\sigma_0$

$= Develop(\rho_\iota'\eta'\theta_t)\chi^*\sigma_0 \hat{} Develop\theta''(\chi^* \dagger (\nu_1+\nu_2))\sigma_2$

where $\nu_1+\nu_2 = \#Develop(\rho_\iota'\eta'\theta_t)\overset{*}{\chi}\sigma_0$

and $\sigma_2 = Consequence(\rho_\iota'\eta'\theta_t)\chi^*\sigma_0$;

hence I_2 is a Terminator for $\rho_\iota', \eta', \chi^*, \sigma_0$.

Hence I^* is a T-List for ρ_ι' ;

so in particular I^* is a T-List for $\rho_1' = \lambda\eta.Run_s(\overset{\iota}{\ast}\rho_\iota\eta)\pi^*$.

QED

Lemma 2.3.2:

If $\langle\rangle$ is a T-List for each ρ_ι

then it is a T-List for $\rho=\lambda\eta.Run_p(\mathop{\#}_{\iota=1}^{\#\rho^*}\rho_\iota\eta)\pi^*$.

Proof

let η' satisfy $\eta'[\![\mathbf{cont}]\!]=\lambda I.\theta_t$

and η satisfy $\eta'=\eta[\lambda I.\theta_t/\mathbf{cont}]$

then $\forall\chi^*,\sigma_0$, since $\langle\rangle$ is a T-List,

$Develop(\rho_\iota\eta\theta_t)\chi^*\sigma_0=Develop(\rho_\iota\eta'\theta_t)\chi^*\sigma_0$;

for each ι,

let $\theta_\iota=\rho_\iota\eta\theta_t$ and $\theta'_\iota=\rho_\iota\eta'\theta_t$

and $\theta^{\pi*}=\langle\pi^*,\lambda\pi_\iota.\theta_\iota\rangle$ and $\theta'^{\pi*}=\langle\pi^*,\lambda\pi_\iota.\theta'_\iota\rangle$;

then $\rho\eta=Combine\theta^{\pi*}\zeta$

and $\rho\eta'=Combine\theta'^{\pi*}\zeta$

where $\zeta=NewSchedule\pi^*$;

so now show by induction on $\nu=\#Develop(Combine\theta^{\pi*}\zeta\theta_t)\chi^*\sigma_0$

that if $\forall\iota,\chi'^*,\sigma$, $Develop\theta_\iota\chi'^*\sigma=Develop\theta'_\iota\chi'^*\sigma$

then $\forall\chi^*,\theta_0,\sigma_0$, $Develop(Combine\theta^{\pi*}\zeta\theta_0)\chi^*\sigma_0$

$= Develop(Combine\theta'^{\pi*}\zeta\theta_0)\chi^*\sigma_0$.

First note that

$(\theta_\iota=\theta_t)\Leftrightarrow(\forall\chi^*,\sigma,\ \#(Develop\theta_\iota\chi^*\sigma)=0)$;

so $(\theta_\iota=\theta_t)\Leftrightarrow(\theta'_\iota=\theta_t)$; (1)

Now consider any χ^*,θ_0,σ_0:

If $AreT\theta^{\pi*}$ then $AreT\theta'^{\pi*}$;

so $\mathcal{L}=\langle\rangle=\mathcal{R}$.

Otherwise:

let $\langle\pi_\iota,\zeta'\rangle=Choose\theta^{\pi*}\zeta$;

then because of (1), $\langle \pi_1, \zeta' \rangle = Choose\theta'^{\pi^*}\zeta$;

let $\langle \chi, \theta \rangle = \theta_1\sigma_0$

and $\langle \chi', \theta' \rangle = \theta'_1\sigma_0$

and $\theta_1^{\pi^*} = \theta^{\pi^*}[\theta/\pi_1]$

and $\theta_1'^{\pi^*} = \theta'^{\pi^*}[\theta'/\pi_1]$;

then $Develop(Combine\theta^{\pi^*}\zeta\theta_0)\chi^*\sigma_0$

$= \langle \chi \rangle \hat{} Develop(Combine\theta_1^{\pi^*}\zeta'\theta_0)(\chi^*\dagger 1)(\chi_1 \circ \chi\sigma_0)$

but $Develop\theta_1\chi^*\sigma_0 = \langle \chi \rangle \hat{} Develop\theta(\chi^*\dagger 1)(\chi_1 \circ \chi\sigma_0)$;

so $\chi = \chi'$;

and $\forall \chi'^*, \sigma_1$, let $\chi_0 = \lambda\sigma.\sigma_1$

then $Develop\theta\chi'^*\sigma_1 = (Develop\theta_1(\langle \chi_0 \rangle \hat{} \chi'^*)\sigma_0)\dagger 1$

$= (Develop\theta'_1(\langle \chi_0 \rangle \hat{} \chi'^*)\sigma_0)\dagger 1$

$= Develop\theta'\chi'^*\sigma_1$;

and $\#Develop(Combine\theta_1^{\pi^*}\zeta'\theta_t)(\chi^*\dagger 1)(\chi_1 \circ \chi\sigma_0) = \nu - 1$;

so by inductive hypothesis,

$Develop(Combine\theta_1^{\pi^*}\zeta'\theta_0)(\chi^*\dagger 1)(\chi_1 \circ \chi\sigma_0)$

$= Develop(Combine\theta_1'^{\pi^*}\zeta'\theta_0)(\chi^*\dagger 1)(\chi_1 \circ \chi\sigma_0)$;

hence $Develop(Combine\theta^{\pi^*}\zeta\theta_0)\chi^*\sigma_0$

$= Develop(Combine\theta'^{\pi^*}\zeta\theta_0)\chi^*\sigma_0$;

Thus $\forall\theta$, $Develop(\rho\eta\theta)\chi^*\sigma_0 = Develop(\rho\eta'\theta)\chi^*\sigma_0$;

but also (cf 2.4.7),

$Develop(\rho\eta'\theta)\chi^*\sigma_0 = Develop(\rho\eta'\theta_t)\chi^*\sigma_0 \hat{} Develop\theta(\chi^*\dagger\nu)\sigma_1$

where $\nu = \#(Develop(\rho\eta'\theta_t)\chi^*\sigma_0)$

and $\sigma_1 = Consequence(\rho\eta'\theta_t)\chi^*\sigma_0$;

so $Develop(\rho\eta\theta)\chi^*\sigma_0 = Develop(\rho\eta'\theta_t)\chi^*\sigma_0 \hat{} Develop\theta(\chi^*\dagger\nu)\sigma_1$;

ie $\checkmark$ is a Terminator for $\rho, \eta', \chi^*, \sigma_0$;

hence $\langle\rangle$ is a T-List for ρ.

QED

Theorem 2.3:

If I^* is a T-List for each ρ_ι

then it is a T-List for $Eval\rho^*$

where $Eval$ may be $Eval_3$

or, if each ρ_ι has an empty T-List, $Eval_4$.

Hence if I^* is a T-List for each $X_\iota[\![\Xi_\iota]\!]$

then it is a T-List also for $XEval[\![\Xi^*]\!]$.

Proof

for $1\leq\iota\leq\#\rho^*$

let $\rho'_\iota=\lambda\eta\theta.\rho_\iota\eta_\iota(\lambda\varepsilon\sigma.\langle Signal\pi_\iota\varepsilon,\theta\rangle)$

where $\eta_\iota=\eta[\pi_\iota/\mathbf{proc}]$;

let I_ι be a Terminator for $\rho_\iota,\eta'_\iota,\chi^*,\sigma_0$

where $\eta'_\iota=\eta'[\pi_\iota/\mathbf{proc}]$;

then $\forall\eta,\theta$, $Develop(\rho'_\iota\eta\theta)\chi^*\sigma_0$

$= Develop(\rho_\iota\eta_\iota(\lambda\varepsilon\sigma.\langle Signal\pi_\iota\varepsilon,\theta\rangle))\chi^*\sigma_0$

$= Develop(\rho_\iota\eta'_\iota\kappa_t)\chi^*\sigma_0\hat{}Develop\theta'(\chi^*\dagger\nu)\sigma_1$

where ν,σ_1 have the usual form

and if $I_\iota=\surd$, $\theta'=(\lambda\sigma.\langle Signal\pi_\iota\varepsilon_\iota,\theta\rangle)$

for some ε_ι ;

if $I_\iota\in\mathbf{Ide}$, $\theta'=\eta_\iota[\![\mathbf{cont}]\!][\![I_1]\!]$

$=\eta[\![\mathbf{cont}]\!][\![I_1]\!]$;

if $I_\iota=?$, $\theta'=\theta_t$ and $ErrorSet\sigma_1$;

so if $I_\iota\neq\surd$,

$Develop(\rho'_\iota\eta'\theta_t)\chi^*\sigma_0=Develop(\rho_\iota\eta'_\iota\kappa_t)\chi^*\sigma_0$

and I_ι is a Terminator for $\rho'_\iota,\eta',\chi^*,\sigma_0$;

if $I_\iota=\surd$,

then $Develop(\rho'_\iota\eta\theta)\chi^*\sigma_0=\chi'^*\hat{}Develop\theta(\chi^*\dagger(\nu+1))\sigma_2$

where $\chi'^*=Develop(\rho_\iota\eta'_\iota\kappa_t)\chi^*\sigma_0\hat{}\langle Signal\pi_\iota\varepsilon_\iota\rangle$;

so again I_ι is a Terminator for $\rho'_\iota,\eta',\chi^*,\sigma_0$.

Hence I^* is a T-List for each ρ'_ι ;

and therefore also for $\rho=\lambda\eta.Run(\rho'_\iota\eta)\pi^*$ (by 2.4.1 and 2.4.2);

so for any η',χ^*,σ_0, let I be a Terminator for $\rho,\eta',\chi^*\dagger 1,\sigma_1$

where $\sigma_1=(\chi_1(CheckProcs\pi\pi^*\sigma_0))$;

then $\forall\eta,\kappa'$, $Develop(Eval\rho^*\eta\kappa')\chi^*\sigma_0$

$=\langle\chi_0\rangle\hat{}Develop(\rho\eta\theta_0)(\chi^*\dagger 1)\sigma_1$

where $\chi_0=CheckProcs\pi\pi^*$

and $\theta_0=(\lambda\sigma.\langle CheckResults\pi^*\varepsilon^*,\kappa'\varepsilon^*\rangle)$

where $\varepsilon^*=Results\pi^*\sigma_0$

$=\langle\chi_0\rangle\hat{}Develop(\rho\eta'\theta_t)(\chi^*\dagger 1)\sigma_1\hat{}Develop\theta'(\chi^*\dagger\nu)\sigma_2$;

if $I\neq\checkmark$ then

$Develop(Eval\rho^*\eta'\kappa'_t)\chi^*\sigma_0=\langle\chi_0\rangle\hat{}Develop(\rho\eta'\theta_t)(\chi^*\dagger 1)\sigma_1$;

and it follows that I is also a Terminator for

$Eval\rho^*,\eta',\chi^*,\sigma_0$;

if $I=\checkmark$ then $Develop(Eval\rho^*\eta\kappa')\chi^*\sigma_0$

$=\langle\chi_0\rangle\hat{}\chi'^*\hat{}\langle\chi'_0\rangle\hat{}Develop(\kappa'\varepsilon^*)(\chi^*\dagger(\nu+1))\sigma_3$

where $\chi'^*=Develop(\rho\eta'\theta_t)(\chi^*\dagger 1)\sigma_1$

and $\chi'_0=CheckResults\pi^*\varepsilon^*$

for some ε^* (independent of κ',η);

and again it immediately follows that

I is a Terminator for $Eval\rho^*,\eta',\chi^*,\sigma_0$.

Hence I^* is a T-List for $Eval\rho^*$.

QED

2.4 Regularity

A useful step in discovering whether Expansions (or Developments) are Miscible is a mechanism for imposing conditions both on any values produced as the result of a computation, and also on the sorts of primitive operations which can be used. This is done by defining the property "Regularity". Conditions on the result of evaluating an expression can obviously be applied only if the expression is successfully evaluated and does actually give a result, so the definition has a form similar to that of JumpFree or Terminator, but its significance lies in the introduction of qualifying predicates.

Definition

$\gamma\epsilon[K\to K]$ is *Regular* wrt τ,τ',τ_1,τ_2 if

$\forall\chi^*,\sigma_0,\varepsilon$ s.t. $\tau_1\varepsilon$ and $\tau^*\chi^*$,

either $\exists\varepsilon'$ s.t. $\tau_2\varepsilon'$ and

$\forall\kappa$, $Develop(\gamma\kappa\varepsilon)\chi^*\sigma_0=\chi'^*\hat{}Develop(\kappa\varepsilon')(\chi^*\dagger\nu)\sigma_1$

and $\tau'^*\chi'^*$

where $\chi'^*=Develop(\gamma\kappa_t\varepsilon)\chi^*\sigma_0$

and $\nu=\#\chi'^*$

and $\sigma_1=Digest\chi'^*\chi^*\sigma_0$

$=Consequence(\gamma\kappa_t\varepsilon)\chi^*\sigma_0$;

or $Develop(\gamma\kappa\varepsilon)\chi^*\sigma_0$ is independent of κ ;

and similarly for $\gamma\epsilon[K\to K']\ [K'\to K]\ [K'\to K']$,

$\gamma\epsilon[C\to K]\ [C\to K']$ wrt τ,τ',τ_1,

$\gamma\epsilon[K\to C]\ [K'\to C]$ wrt τ,τ',τ_2,

and $\gamma\epsilon[C\to C]$ wrt τ,τ'.

The predicates imposed on state transformations will usually have the effects of

a) restricting the processes whose results the operation can change;

b) ensuring that certain things, such as the discovery of an error or the claiming of a new location are not "forgotten".

Thus they will usually take one of the forms:

$$\mathit{Invariant}\pi^* =$$
$$\lambda\chi.(\forall\pi\sigma,((\exists\pi_1 \lhd \pi^*,\leq\pi) \Rightarrow (\mathit{Result}\pi(\chi\sigma)=\mathit{Result}\pi\sigma)) \wedge$$
$$\forall\sigma,(\mathit{ErrorSet}\sigma \Rightarrow \mathit{ErrorSet}(\chi\sigma)) \wedge$$
$$\forall\alpha\sigma,(\sigma[\![\mathbf{area}]\!]\alpha \Rightarrow (\chi\sigma)[\![\mathbf{area}]\!]\alpha))$$

$$\mathit{Subvariant}\pi^* =$$
$$\lambda\chi.(\forall\pi\sigma,(\sim(\exists\pi_1 \lhd \pi^*,<\pi) \Rightarrow (\mathit{Result}\pi(\chi\sigma)=\mathit{Result}\pi\sigma)) \wedge$$
$$\forall\sigma,(\mathit{ErrorSet}\sigma \Rightarrow \mathit{ErrorSet}(\chi\sigma)) \wedge$$
$$\forall\alpha\sigma,(\sigma[\![\mathbf{area}]\!]\alpha \Rightarrow (\chi\sigma)[\![\mathbf{area}]\!]\alpha))$$

Note that both $\mathit{Invariant}\pi^*\chi$ and $\mathit{Subvariant}\pi^*\chi$ imply

$$\forall\sigma(\mathit{Results}\pi^*(\chi\sigma)=\mathit{Results}\pi^*\sigma) \ .$$

The predicates imposed on expression values will often depend on the environment, and in order that *Eval* preserves Regularity with respect to such predicates, they will have to be constrained to satisfy the following condition:

Definition

The predicates τ_η are *H-Monotonic* if

for any η, $\pi' \geq \eta[\![\mathbf{proc}]\!]$ and ε

if $\eta' = \eta[\pi'/\mathbf{proc}]$

then $\tau_{\eta'}\varepsilon \Rightarrow \tau_\eta\varepsilon$.

Before embarking on results about the preservation of Regularity by *Eval*, it is appropriate to include a lemma for later use: the Regularity condition provides a useful basis for the proposition (promised in 2.1) that, under suitable conditions, *Detail* is a special case of *Develop*.

Lemma 2.4.1:

If ρ^* satisfies:

for any η', each $\rho_\iota\eta'$ is Regular wrt

(*Invariant*$\langle \eta'[\![\mathbf{proc}]\!] \rangle$), (*Subvariant*$\langle \eta'[\![\mathbf{proc}]\!] \rangle$) ;

and $\theta^{\pi*} = \langle \pi^*, \lambda\pi_\iota.\rho_\iota(\eta[\pi_\iota/\mathbf{proc}])\theta_t \rangle$

then for each ι,

$\exists \chi^*, \chi'$ s.t. (*Invariant*$\langle \pi_\iota \rangle$)$^*\chi^*$ and *Invariant*$\langle \pi_\iota \rangle\, \chi'$

and *Detail*$\theta^{\pi*}\zeta\pi_\iota\sigma$ = *Develop*$(\theta^{\pi*}\pi_\iota)\chi^*(\chi'\sigma)$.

Proof

For each ι, let τ_ι denote (*Invariant*$\langle \pi_\iota \rangle$)

and τ'_ι denote (*Subvariant*$\langle \pi_\iota \rangle$) ;

The Regularity condition implies

if $\tau_\iota^*\chi'^*$ then τ'^*_ι(*Develop*$(\theta^{\pi*}\pi_\iota)\chi'^*\sigma$) ; (1)

So for any $\theta^{\pi*}$ and σ satisfying (1) and for any ι,

proceed by induction on $\nu = \#$*Detail*$\theta^{\pi*}\zeta\pi_\iota\sigma$.

If $\theta^{\pi*}\pi_\iota = \theta_t$ then $\mathcal{L} = \langle\rangle = \mathcal{R}$.

Otherwise induct also on ι_0 = least integer s.t. $\zeta\downarrow\iota_0=\pi_\iota$:

let $\langle \pi',\zeta'\rangle = Choose\theta^{\pi*}\zeta$

$= \langle \zeta\downarrow\iota',\zeta\dagger\iota'\rangle$ for some $\iota'\leq\iota_0$

and $\langle \chi,\theta'\rangle = \theta^{\pi*}\pi'\sigma$

and $\theta'^{\pi*}=\theta^{\pi*}[\theta'/\pi']$;

then $\theta'^{\pi*}$ and $(\chi\sigma)$ satisfy (1) ;

for if $\pi_\iota\neq\pi'$

then $Develop(\theta'^{\pi*}\pi_\iota)\chi'^*(\chi\sigma)=Develop(\theta^{\pi*}\pi_\iota)\chi'^*(\chi\sigma)$;

and if $\pi_\iota=\pi'$

then $Develop(\theta'^{\pi*}\pi_\iota)\chi'^*(\chi\sigma)=$

$(Develop(\theta^{\pi*}\pi_\iota)(\langle\lambda\sigma.\sigma\rangle\hat{}\chi'^*)\sigma)\dagger 1$;

so if $\pi_\iota=\pi'$

then $\mathcal{L} = Detail\theta^{\pi*}\zeta\pi_\iota\sigma$

$= \langle\chi\rangle\hat{}Detail\theta'^{\pi*}\zeta'\pi_\iota(\chi\sigma)$

$= \langle\chi\rangle\hat{}Develop(\theta'^{\pi*}\pi_\iota)\chi^*(\chi'\circ\chi\sigma)$

(by ν inductive hypothesis)

for some χ^*,χ' s.t. $\tau^*\chi^*$ and $\tau\chi'$

$= Develop(\theta^{\pi*}\pi_\iota)(\langle\chi'\rangle\hat{}\chi^*)\sigma$;

and if $\pi_\iota\neq\pi'$

then $\mathcal{L} = Detail\theta^{\pi*}\zeta\pi_\iota\sigma$

$= Detail\theta'^{\pi*}\zeta'\pi_\iota(\chi\sigma)$;

but also $1\leq\iota'<\iota_0$, so $\zeta'\downarrow(\iota_0-\iota')=\pi_\iota$;

hence by ι_0 inductive hypothesis,

$\mathcal{L} = Develop(\theta'^{\pi*}\pi_\iota)\chi^*(\chi'\circ\chi\sigma)$

for some χ^*,χ' as above

$= Develop(\theta^{\pi*}\pi_\iota)\chi^*(\chi''\sigma)$

where $\chi''=\chi'\circ\chi$;

and since by (1) $Subvariant\langle\pi'\rangle\chi$, and so $\tau\chi$;

so $\tau\chi''$.

QED

Two more lemmas, which will be needed later when finally assembling the different properties to prove the equivalence of serial and parallel execution, show that the serial forms (only) of *Run* and *Eval* preserve the $\approx$ equivalence of Expansions. Note that the parallel versions do not have this property since the interleaving of different processes' operations could, as mentioned in 1.2, produce different operations.

Although they have little relation to the rest of this section, it is convenient to include these lemmas here, as they depend on Regularity. Also, one of them depends on results proved further on in this section, and so should logically come later, but it appears here in the interest of getting the minor results out of the way before starting on the major work of the section.

Lemma 2.4.2:

If $\gamma^*,\gamma'^* \in [C \to C]$ satisfy $\#\gamma^* = \#\gamma'^*$

and for each ι,

γ_ι and γ'_ι are Regular wrt τ,τ'

for some τ,τ' satisfying $\tau(\lambda\sigma.\sigma)$

and $\forall\theta,\sigma$, $Expand(\gamma_\iota\theta)\sigma \approx Expand(\gamma'_\iota\theta)\sigma$

then $\forall\pi^*,\theta,\sigma$, $Expand(Run_S\gamma^*\pi^*\theta)\sigma \approx Expand(Run_S\gamma'^*\pi^*\theta)\sigma$.

Proof

let $\psi = Fix(\lambda\psi.\lambda\iota\theta.\iota \le \#\gamma^* \to \gamma_\iota(\psi(\iota+1)\theta),\ \theta)$

and $\psi' = Fix(\lambda\psi'.\lambda\iota\theta.\iota \le \#\gamma'^* \to \gamma'_\iota(\psi'(\iota+1)\theta),\ \theta)$

then it will be shown (by induction on $\#\gamma^* - \iota$) that

$$Expand(\psi\iota\theta)\sigma \approx Expand(\psi'\iota\theta)\sigma \qquad (1)\ .$$

If $\iota > \#\gamma^*$ then $\mathcal{L} = \mathit{Expand}\theta\sigma = \mathcal{R}$.

If $\iota \leq \#\gamma^*$

then $\mathcal{L} = \mathit{Expand}(\gamma_\iota(\psi(\iota+1)\theta))\sigma$;

but by Regularity of γ_ι,

either $\forall\sigma$, $\mathit{Expand}(\gamma_\iota\theta)\sigma = \mathit{Expand}(\gamma_\iota\theta_t)\sigma \hat{} \mathit{Expand}\theta\sigma_1$

or $\forall\sigma$, $\mathit{Expand}(\gamma_\iota\theta)\sigma = \mathit{Expand}(\gamma_\iota\theta_t)\sigma$

where $\sigma_1 = \mathit{Outcome}(\gamma_\iota\theta_t)\sigma$;

and similarly for γ'_ι, with the same σ_1.

Furthermore, the same case holds for both γ_ι and γ'_ι ;

since otherwise eg,

$$\begin{aligned}\forall\theta,\ \mathit{Expand}(\gamma_\iota\theta_t)\sigma &\approx \mathit{Expand}(\gamma'_\iota\theta_t)\sigma \\ &\approx \mathit{Expand}(\gamma'_\iota\theta)\sigma \\ &\approx \mathit{Expand}(\gamma_\iota\theta)\sigma \\ &\approx \mathit{Expand}(\gamma_\iota\theta_t)\sigma \hat{} \mathit{Expand}\theta\sigma_1 \ ;\end{aligned}$$

hence $\#\mathit{Expand}\theta\sigma_1 = 0$ ie $\theta = \theta_t$;

so either $\mathcal{L} = \mathit{Expand}(\gamma_\iota\theta_t)\sigma \hat{} \mathit{Expand}(\psi(\iota+1)\theta)\sigma_1$

$\approx \mathit{Expand}(\gamma'_\iota\theta_t)\sigma \hat{} \mathit{Expand}(\psi'(\iota+1)\theta)\sigma_1$

(using inductive hypothesis)

$\approx \mathit{Expand}(\gamma'_\iota(\psi'(\iota+1)\theta))\sigma$

$\approx \mathcal{R}$;

or $\mathcal{L} = \mathit{Expand}(\gamma_\iota\theta_t)\sigma$

$\approx \mathit{Expand}(\gamma'_\iota\theta_t)\sigma$

$\approx \mathit{Expand}(\gamma'_\iota(\psi'(\iota+1)\theta))\sigma$

$\approx \mathcal{R}$;

hence (1) holds for all $\iota \geq 1$;

putting $\iota = 1$ gives desired result.

QED

Lemma 2.4.3:

If $\rho^*, \rho'^*, \eta_0, \eta_0'$ satisfy $\#\rho^* = \#\rho'^*$ and $\eta_0[\![\mathbf{proc}]\!] = \eta_0'[\![\mathbf{proc}]\!]$

and for each ι, $\forall \eta, \eta'$, $\rho_\iota \eta$ and $\rho_\iota' \eta'$ are Regular

(wrt some τ, τ', τ_ι s.t. $\tau(\lambda\sigma.\sigma)$)

and $\forall \kappa, \sigma, \pi_\iota$, $Expand(\rho_\iota \eta_\iota \kappa)\sigma \approx Expand(\rho_\iota' \eta_\iota' \kappa)\sigma$

where $\eta_\iota = \eta_0[\pi_\iota/\mathbf{proc}]$ and $\eta_\iota' = \eta_0'[\pi_\iota/\mathbf{proc}]$

then $\forall \kappa', \sigma$, $Expand(Eval_3 \rho^* \eta_0 \kappa')\sigma \approx Expand(Eval_3 \rho'^* \eta_0' \kappa')\sigma$.

Proof

let $\pi = \eta_0[\![\mathbf{proc}]\!]$

and $\pi^* = NewProcs\,\pi(\#\rho^*)\sigma$;

let $\gamma_\iota = \lambda\theta.\rho_\iota \eta_\iota(\lambda\varepsilon\sigma.\langle Signal\,\pi_\iota \varepsilon, \theta\rangle)$;

but $\lambda\theta\varepsilon\sigma.\langle Signal\,\pi_\iota \varepsilon, \theta\rangle$ is Regular wrt suitable predicates;

so γ_ι is Regular and hence (by 2.4.6) $Run_s \gamma^* \pi^*$ is Regular;

and similarly for γ_ι' etc.

Furthermore, as in 2.4.2, ρ_ι and ρ_ι' satisfy the same

alternative of the Regularity condition.

so either $\exists \varepsilon$ s.t.

$$\begin{aligned} Expand(\gamma_\iota \theta)\sigma &= Expand(\rho_\iota \eta_\iota(\lambda\varepsilon\sigma.\langle Signal\,\pi_\iota \varepsilon, \theta\rangle))\sigma \\ &= Expand(\rho_\iota \eta_\iota \kappa_t)\sigma \hat{}\langle Signal\,\pi_\iota \varepsilon\rangle \hat{}\, Expand\,\theta\sigma_1 \\ &\approx Expand(\rho_\iota' \eta_\iota' \kappa_t)\sigma \hat{}\langle Signal\,\pi_\iota \varepsilon\rangle \hat{}\, Expand\,\theta\sigma_1' \\ &\approx Expand(\gamma_\iota' \theta)\sigma \ ; \end{aligned}$$

since $\sigma_1 = Signal\,\pi_\iota \varepsilon(Outcome(\rho_\iota \eta_\iota \kappa_t)\sigma)$

$= Signal\,\pi_\iota \varepsilon(Outcome(\rho_\iota' \eta_\iota' \kappa_t)\sigma)$

$= \sigma_1'$;

or $Expand(\gamma_l\theta)\sigma = Expand(\rho_l\eta_l\kappa_t)\sigma$

$\approx Expand(\rho'_l\eta'_l\kappa_t)\sigma$

$\approx Expand(\gamma'_l\theta)\sigma$;

hence by 2.4.2,

$Expand(Run_s\gamma^*\pi^*\theta)\sigma \approx Expand(Run_s\gamma'^*\pi^*\theta)\sigma$;

again, this implies that $Run_s\gamma^*\pi^*$ and $Run_s\gamma'^*\pi^*$ satisfy the same alternative of Regularity;

ie either $Expand(Eval_3\rho^*\eta_0\kappa')\sigma_0 = \langle\chi_0\rangle\hat{}Expand(Run_s\gamma^*\pi^*\theta_t)\sigma_1\hat{}$
$Expand\theta_0\sigma_2$

$\approx \langle\chi_0\rangle\hat{}Expand(Run_s\gamma'^*\pi^*\theta_t)\sigma_1\hat{}$
$Expand\theta_0\sigma_2$

$\approx Expand(Eval_3\rho'^*\eta'_0\kappa')\sigma_0$

where $\chi_0 = CheckProcs\pi\pi^*$

$\sigma_1 = \chi_0\sigma_0$

$\sigma_2 = Outcome(Run_s\gamma^*\pi^*\theta_t)\sigma_1$

and $\theta_0 = UnlessError(...)$;

or $Expand(Eval_3\rho^*\eta_0\kappa')\sigma_0 = \langle\chi_0\rangle\hat{}Expand(Run_s\gamma^*\pi^*\theta_t)\sigma_1$

$\approx \langle\chi_0\rangle\hat{}Expand(Run_s\gamma'^*\pi^*\theta_t)\sigma_1$

$\approx Expand(Eval_3\rho'^*\eta'_0\kappa')\sigma_0$.

QED

The next few lemmas show that the property of Regularity can be combined with that of JumpFree with the expected effect, and is preserved by function composition and by both forms of *Run*.

Lemma 2.4.4:

If $\gamma\in[K\to K]$ is Regular wrt τ,τ',τ_1,τ_2 and JumpFree
then $\forall\chi^*,\sigma_0,\varepsilon$ satisfying $\tau_1\varepsilon$ and $\tau^*\chi^*$,
either $\exists\varepsilon'$ s.t. $\tau_2\varepsilon'$ and
$$Develop(\gamma\kappa\varepsilon)\chi^*\sigma_0 = \chi'^*\hat{}Develop(\kappa\varepsilon')(\chi^*\dagger\nu)\sigma_1$$
with χ'^*,ν,σ_1 as in definition of Regular,
or $Develop(\gamma\kappa\varepsilon)\chi^*\sigma_0$ is independent of κ
and $ErrorSet\sigma_1$.

Proof

Take any $\chi^*,\sigma_0,\varepsilon$ s.t. $\tau^*\chi^*$ and $\tau_1\varepsilon$.
If the first alternative of Regularity is satisfied
then there is nothing more to prove;
Similarly if the second alternative of Regularity
and the second of JumpFree are satisfied.
So suppose the first alternative of Regularity
and the second of JumpFree:
then $\forall\kappa_1,\kappa_2$, $Develop(\gamma\kappa_1\varepsilon)\chi^*\sigma_0=Develop(\gamma\kappa_2\varepsilon)\chi^*\sigma_0$;
but $\exists\varepsilon'$ s.t. for $\iota=1,2$,
$$Develop(\gamma\kappa_\iota\varepsilon)\chi^*\sigma_0=Develop(\gamma\kappa_t\varepsilon)\chi^*\sigma_0\hat{}Develop(\kappa_\iota\varepsilon')(\chi^*\dagger\nu)\sigma_1 ;$$
hence $Develop(\kappa_1\varepsilon')(\chi^*\dagger\nu)\sigma_1=Develop(\kappa_2\varepsilon')(\chi^*\dagger\nu)\sigma_1$;
in particular,
$$Develop(\kappa_1\varepsilon')(\chi^*\dagger\nu)\sigma_1=Develop(\kappa_t\varepsilon')(\chi^*\dagger\nu)\sigma_1=() ;$$
which implies $\kappa_1=\kappa_t$;
contradicting arbitrary choice of κ_1.

QED

Lemma 2.4.5:

If $\gamma_1 \in [K \to K]$ is Regular wrt $\tau, \tau', \tau_1, \tau_2$
and $\gamma_2 \in [K \to K]$ is Regular wrt $\tau, \tau', \tau_2, \tau_3$
then $\gamma_1 \circ \gamma_2$ is Regular wrt $\tau, \tau', \tau_1, \tau_3$;

and similarly for other possible functionalities of γ_1, γ_2.

Proof

Consider any $\chi^*, \sigma_0, \varepsilon$ s.t. $\tau^* \chi^*$ and $\tau_1 \varepsilon$;
let $\chi_1^* = Develop(\gamma_1 \kappa_t \varepsilon) \chi^* \sigma_0$;
then either $\exists \varepsilon'$ s.t. $\tau_2 \varepsilon'$ and

$\forall \kappa$, $Develop(\gamma_1(\gamma_2 \kappa)\varepsilon)\chi^*\sigma_0 = \chi_1^* \hat{} Develop(\gamma_2 \kappa \varepsilon')\chi'^*\sigma_1$
where $\chi'^* = \chi^* \dagger (\# \chi_1^*)$
and $\sigma_1 = Consequence(\gamma_1 \kappa_t \varepsilon)\chi^*\sigma_0$
and $\tau'^* \chi_1^*$;

now let $\chi_2^* = Develop(\gamma_2 \kappa_t \varepsilon')\chi'^*\sigma_1$;
either $\exists \varepsilon''$ s.t. $\tau_3 \varepsilon''$ and

$\forall \kappa$, $Develop(\gamma_2 \kappa \varepsilon')\chi'^*\sigma_1 = \chi_2^* \hat{} Develop(\kappa \varepsilon'')\chi''^*\sigma_2$
where $\chi''^* = \chi'^* \dagger (\# \chi_2^*)$
and $\sigma_2 = Consequence(\gamma_2 \kappa_t \varepsilon')\chi'^*\sigma_1$
and $\tau'^* \chi_2^*$;

so in particular, $Develop(\gamma_1(\gamma_2 \kappa_t)\varepsilon)\chi^*\sigma_0 = \chi_1^* \hat{} \chi_2^*$
and $\sigma_2 = Digest \chi_2^* \chi'^* (Digest \chi_1^* \chi^* \sigma_0)$
$= Digest(\chi_1^* \hat{} \chi_2^*)\chi^*\sigma_0$
$= Consequence(\gamma_1(\gamma_2 \kappa_t)\varepsilon)\chi^*\sigma_0$;
so $\forall \kappa$,

$Develop(\gamma_1(\gamma_2 \kappa)\varepsilon)\sigma_0 = (\chi_1^* \hat{} \chi_2^*) \hat{} Develop(\kappa \varepsilon'')\chi''^*\sigma_2$
where $\chi''^* = \chi^* \dagger \# (\chi_1^* \hat{} \chi_2^*)$
and $\tau'^* (\chi_1^* \hat{} \chi_2^*)$;

or $Develop(\gamma_2 \kappa \varepsilon')\chi'^* \sigma_1$ is independent of κ ;

ie $\forall \kappa_1, \kappa_2$, $Develop(\gamma_1(\gamma_2 \kappa_1)\varepsilon)\chi^* \sigma_0$

$= \chi_1^* \hat{} Develop(\gamma_2 \kappa_1 \varepsilon')\chi'^* \sigma_1$

$= \chi_1^* \hat{} Develop(\gamma_2 \kappa_2 \varepsilon')\chi'^* \sigma_1$

$= Develop(\gamma_1(\gamma_2 \kappa_2)\varepsilon)\chi^* \sigma_0$

and $Develop(\gamma_1(\gamma_2 \kappa_t)\varepsilon)\chi^* \sigma_0 = \chi_1^* \hat{} \chi_2^*$;

so $\tau'^*(Develop(\gamma_1(\gamma_2 \kappa_t)\varepsilon)\chi^* \sigma_0)$;

or $Develop(\gamma_1 \kappa \varepsilon)\chi^* \sigma_0$ is independent of κ ;

so $\forall \kappa_1, \kappa_2$, $Develop(\gamma_1(\gamma_2 \kappa_1)\varepsilon)\chi^* \sigma_0 = Develop(\gamma_1(\gamma_2 \kappa_2)\varepsilon)\chi^* \sigma_0$

and $Develop(\gamma_1(\gamma_2 \kappa_t)\varepsilon)\chi^* \sigma_0 = Develop(\gamma_1 \kappa_t \varepsilon)\chi^* \sigma_0 = \chi_1^*$;

so $\tau'^*(Develop(\gamma_1(\gamma_2 \kappa_t)\varepsilon)\chi^* \sigma_0)$.

QED

Lemma 2.4.6:

If $\gamma^* \in [C \to C]^*$

and each γ_ι is Regular wrt τ, τ'

then $Run_S \gamma^* \pi^*$ is Regular wrt τ, τ'.

Proof

let $\psi = Fix(\lambda \psi . \lambda \iota \theta . \iota \leq \# \gamma^* \to \gamma_\iota(\psi(\iota+1)\theta), \theta)$;

then (by induction on $\# \gamma^* - \iota$) $\psi \iota$ is Regular wrt τ, τ'.

for if $\iota > \# \gamma^*$ then $\psi \iota = \lambda \theta . \theta$ which is trivially Regular;

and if $\iota \leq \# \gamma^*$ then $\psi \iota = \gamma_\iota \circ \psi(\iota+1)$;

but γ_ι is Regular wrt τ, τ' by hypothesis;

and $\psi(\iota+1)$ is Regular by inductive hypothesis;

so by 2.4.5, $\psi \iota$ is Regular wrt τ, τ' ;

in particular, $Run_S\gamma^*\pi^*=\psi 1$ is Regular wrt τ,τ'.

QED

Lemma 2.4.7:

If τ,τ' satisfy

$\forall\chi^*,\sigma_0,\ \tau^*\chi^* \Rightarrow \tau'^*(Develop\theta_\iota\chi^*\sigma_0)$

and $\theta^{\pi*}=\langle \pi^*,\lambda\pi_\iota.\theta_\iota\rangle$

then $Combine\theta^{\pi*}\zeta$ is Regular wrt τ,τ' and JumpFree;

hence $Run_P\gamma^*\pi^*$ is Regular wrt τ,τ' and JumpFree.

Proof

By induction on ν ($\nu=\#Develop(\gamma\theta_t)\chi^*\sigma_0$ and $\gamma=Combine\theta^{\pi*}\zeta$), the following slightly stronger property wll be proved:

$\forall\theta,\ Develop(\gamma\theta)\chi^*\sigma_0=\chi_1^*\hat{}Develop\theta(\chi^*\dagger\nu)\sigma_1$

and $\tau'^*\chi_1^*$

(with χ_1^* and σ_1 as in definition of Regular).

If $AreT\theta^{\pi*}$

then $\gamma=\lambda\theta.\theta$, which trivially satisfies the condition since $\nu=0$ and $\sigma_1=\sigma_0$.

Otherwise:

let $\langle \pi,\zeta'\rangle=Choose\theta^{\pi*}\zeta$

and $\langle \chi,\theta'\rangle=\theta^{\pi*}\pi\sigma_0$;

also let $\theta'^{\pi*}=\theta^{\pi*}[\theta'/\pi]$

and $\gamma'=Combine\theta'^{\pi*}\zeta'$;

then $\forall\theta,\ \gamma\theta\sigma_0=\langle \chi,\gamma'\theta\rangle$;

so $Develop(\gamma\theta)\chi^*\sigma_0=\langle \chi\rangle\hat{}Develop(\gamma'\theta)(\chi^*\dagger 1)(\chi_1\circ\chi\sigma_0)$;

so $\#Develop(\gamma'\theta)(\chi^*\dagger 1)(\chi_1\circ\chi\sigma_0)=\nu-1$;

so by inductive hypothesis,

$Develop(\gamma'\theta)(\chi^*\dagger 1)(\chi_1\circ\chi\sigma_0) = \chi_1'^*\hat{}Develop\theta(\chi^*\dagger 1\dagger(\nu-1))\sigma_1$

and $\tau'^{*}\chi_1'^{*}$

where $\chi_1'^{*}=Develop(\gamma'\theta_t)(\chi^{*}\dagger 1)(\chi_1\circ\chi\sigma_0)$

and $\sigma_1=Consequence(\gamma'\theta_t)(\chi^{*}\dagger 1)(\chi_1\circ\chi\sigma_0)$

$=Consequence(\gamma\theta_t)\chi^{*}\sigma_0$;

so $Develop(\gamma\theta)\chi^{*}\sigma_0 = \langle\chi\rangle\hat{}\chi_1'^{*}\hat{}Develop\theta(\chi^{*}\dagger\nu)\sigma_1$

$= \chi'^{*}\hat{}Develop\theta(\chi^{*}\dagger\nu)\sigma_1$

where $\chi'^{*}=Develop(\gamma\theta_t)\chi^{*}\sigma_0$;

also $\chi\prec Develop(\theta^{\pi *}\pi)\chi^{*}\sigma_0$ so $\tau'\chi$;

hence $\tau'^{*}\chi'^{*}$

QED

The next stage is to extend the results about *Run* to apply also to *Eval*; but this is considerably complicated by *Eval*'s use of the **res** component of the state to keep results from processes: it is clearly essential to be certain that the values produced by a set of offspring processes are the same as those supplied to the parent process to continue its computation. The next two lemmas show that both versions of *Run* cause essentially the correct results to be left. The preservation of Regularity by *Eval* follows. Note that, since $Eval_4$ is only for application to objects satisfying a P-Test, and which are therefore JumpFree, 2.4.9 can hypothesise JumpFreeness as well as Regularity.

Lemma 2.4.8:

If τ,τ' satisfy

$\forall\chi,\ \tau\chi\vee\tau'\chi \Rightarrow \forall\sigma,(Results\pi^{*}(\chi\sigma)=Results\pi^{*}\sigma)$

and each γ_l is Regular wrt τ,τ' and some τ_l

and $\gamma_l'=\lambda\theta.(\lambda\varepsilon\sigma.\langle Signal\pi_l\varepsilon,\theta\rangle)$

and $\gamma_l''=\gamma_l\circ\gamma_l'$

then $\gamma=(Run_s\gamma''^*\pi^*\circ UnlessError)$ satisfies

$\forall\chi^*,\sigma_0$ s.t. $\tau^*\chi^*$,

either $\forall\theta$, $Develop(\gamma\theta)\chi^*\sigma_0=\chi'^*\hat{}Develop\theta(\chi^*\dagger\nu)\sigma_1$

with χ'^*,ν,σ_1 as in definition of Regular

and for each ι, $\tau_\iota(Result\pi_\iota\sigma_1)$

or $Develop(\gamma\theta)\chi^*\sigma_0$ is independent of θ.

Proof

let $\psi=Fix(\lambda\psi.\lambda\iota\theta.\iota\leq\#\gamma^*\rightarrow\gamma''_\iota(\psi(\iota+1)\theta),\theta)$;

then show by induction (on $\#\gamma^*-\iota$) that $\psi\iota$ satisfies: (1)

$\forall\chi^*,\sigma_0$ s.t. $\tau^*\chi^*$

either $\forall\theta$, $Develop(\psi\iota\theta)\chi^*\sigma_0=\chi'^*\hat{}Develop\theta(\chi^*\dagger\nu)\sigma_1$

and for $1\leq\iota'\leq\iota-1$ $Result\pi_{\iota'},\sigma_1=Result\pi_{\iota'},\sigma_0$

and for $\iota\leq\iota'\leq\#\gamma^*$, $\tau_{\iota'},(Result\pi_{\iota'},\sigma_1)$

or $Develop(\psi\iota\theta)\chi^*\sigma_0$ is independent of θ .

If $\iota=\#\gamma^*+1$ then $\psi\iota\theta_t=\theta_t$;

so $\sigma_1=\sigma_0$;

and condition on results becomes

for $1\leq\iota'\leq\#\gamma^*$, $Result\pi_{\iota'},\sigma_1=Result\pi_{\iota'},\sigma_0$,

which is trivially true.

If $\iota\leq\#\gamma^*$

then $\psi\iota=\gamma_\iota\hat{}\gamma'_\iota\circ\psi(\iota+1)$;

so, by Regularity of γ_ι, for any χ^*,σ_0 s.t. $\tau^*\chi^*$,

either $\exists\varepsilon$ s.t. $\tau_\iota\varepsilon$ and

$$\forall\theta,\ Develop(\psi\iota\theta)\chi^*\sigma_0 = \chi_\iota^*\hat{}Develop(\gamma'_\iota(\psi(\iota+1)\theta)\varepsilon)(\chi^*\dagger\nu_1)\sigma_1$$
$$= \chi_\iota^*\hat{}(\chi_\iota)\hat{}Develop(\psi(\iota+1)\theta)(\chi^*\dagger\nu_2)\sigma_2$$

where $\chi_\iota^*=Develop(\gamma_\iota\kappa_t)\chi^*\sigma_0$

and $\chi_\iota=Signal\pi_\iota\varepsilon$

and $\nu_1=\#\chi_\iota^*$

and $\nu_2=\nu_1+1$

and $\sigma_1=Consequence(\gamma_\iota\kappa_t)\chi^*\sigma_0$

and $\sigma_2=\chi_\iota\sigma_1$;

but by inductive hypothesis, $\psi(\iota+1)$ satisfies (1);

so either

$\forall\theta$, $Develop(\psi\iota\theta)\chi^*\sigma_0=\chi_\iota^*\hat{}\langle\chi_\iota^*\rangle\hat{}\chi''^*\hat{}Develop\theta(\chi^*\dagger\nu_3)\sigma_3$

where $\chi''^*=Develop(\psi(\iota+1)\theta_t)(\chi^*\dagger\nu_2)\sigma_2$

and for $1\leq\iota'\leq\iota$, $Result\pi_\iota,\sigma_3=Result\pi_\iota,\sigma_2$

and for $\iota+1\leq\iota'\leq\#\gamma^*$, $\tau_\iota,(Result\pi_\iota,\sigma_3)$;

so, substituting for θ_t,

$\forall\theta$, $Develop(\psi\iota\theta)\chi^*\sigma_0=\chi'^*\hat{}Develop\theta(\chi^*\dagger\nu_3)\sigma_3$

where χ'^*,ν_3,σ_3 satisfy the conditions of (1);

and for $1\leq\iota'\leq\iota-1$, $Result\pi_\iota,\sigma_3 = Result\pi_\iota,\sigma_2$

$= Result\pi_\iota,\sigma_1$

$= Result\pi_\iota,\sigma_0$

since $\tau'^*(\chi_\iota^*)$;

and $Result\pi_\iota\sigma_3 = Result\pi_\iota\sigma_2 = \varepsilon$

so $\tau_\iota(Result\pi_\iota\sigma_3)$

and for $\iota+1\leq\iota'\leq\#\gamma^*$, $\tau_\iota,(Result\pi_\iota,\sigma_3)$;

or $\forall\theta_1,\theta_2$, $Develop(\psi\iota\theta_1)\chi^*\sigma_0$

$= \chi_\iota^*\hat{}\langle\chi_\iota\rangle\hat{}Develop(\psi(\iota+1)\theta_1)(\chi^*\dagger\nu_2)\sigma_2$

$= \chi_\iota^*\hat{}\langle\chi_\iota\rangle\hat{}Develop(\psi(\iota+1)\theta_2)(\chi^*\dagger\nu_2)\sigma_2$

$- Develop(\psi\iota\theta_2)\chi^*\sigma_0$;

or $\forall\theta_1,\theta_2$, $Develop(\psi\iota\theta_1)\chi^*\sigma_0$ $Develop(\gamma_\iota(\gamma'_\iota(\psi(\iota+1)\theta_1)))\chi^*\sigma_0$

$= Develop(\gamma_\iota(\gamma'_\iota(\psi(\iota+1)\theta_2)))\chi^*\sigma_0$

$= Develop(\psi\iota\theta_2)\chi^*\sigma_0$;

Hence $\psi\iota$ satisfies (1) for $1\leq\iota\leq\#\gamma^*+1$;

Putting $\iota=1$ gives required form for $Run_s\gamma''^*\pi^*$;

so for any χ^*,σ_0 s.t. $\tau^*\chi^*$,

either $\forall\theta$ $Develop(\gamma\theta)\chi^*\sigma_0 = Develop(\psi 1\theta_t)\chi^*\sigma_0\,\hat{}$

$Develop(UnlessError\theta)(\chi^*\dagger\nu)\sigma_1$

and for all ι, $\tau_\iota(Result\pi_\iota\sigma_1)$;

but if $ErrorSet\sigma_1$

then $Develop(UnlessError\theta)(\chi^*\dagger\nu)\sigma_1$ =

$\langle CheckError\ True\rangle$;

so $Develop(\gamma\theta)\chi^*\sigma_0$ is independent of θ ;

otherwise:

$Develop(UnlessError\theta)(\chi^*\dagger\nu)\sigma_1$ =

$\langle CheckError\ False\rangle\,\hat{}\,Develop\theta(\chi^*\dagger\nu+1)\sigma_2$

where $\sigma_2 = CheckError\ False\sigma_1 = \sigma_1$;

so $\forall\theta$, $Develop(\gamma\theta)\chi^*\sigma_0 = \chi'^*\,\hat{}\,Develop\theta(\chi^*\dagger\nu')\sigma_1$

where χ'^*,ν',σ_1 are as in (1);

and for all ι, $\tau_\iota(Result\pi_\iota\sigma_1)$;

or $Develop(\psi\iota\theta)\chi^*\sigma_0$ is independent of θ;

hence so is $Develop(\gamma\theta)\chi^*\sigma_0$.

QED

Lemma 2.4.9:

If the hypotheses of 2.4.8 are satisfied and also

$\forall\chi$, $\tau\chi\vee\tau'\chi \Rightarrow (\forall\sigma,\ ErrorSet\sigma \Rightarrow ErrorSet(\chi\sigma))$

and each γ_ι is JumpFree

then $(Run_p\gamma''^*\pi^*\circ UnlessError)$

satisfies the conclusions of 2.4.8.

Proof

For any θ^*,π^*,ζ fair in π^*,

let $\theta^{\pi*}=\langle \pi^*,\lambda\pi_\iota.\theta_\iota\rangle$

and $\gamma=Combine\theta^{\pi*}\zeta$;

and first show by induction on $\nu=\#Develop(\gamma\theta_t)\chi^*\sigma$ that

if each θ_ι satisfies: (1)

for any σ,χ^*,

if $\chi'^*=Develop\theta_\iota\chi^*\sigma$

then $\forall\chi\prec\chi'^*,\forall\pi\prec(\pi^*\setminus\langle\pi_\iota\rangle),\forall\sigma',Result\pi(\chi\sigma')=Result\pi\sigma'$

and either if $\#\chi'^*>0$

then $\forall\sigma'$, $\tau_\iota(Result\pi_\iota((\chi'^*\downarrow(\#\chi'^*))\sigma'))$

or $ErrorSet(Consequence\theta_\iota\chi^*\sigma)$;

then $\forall\sigma,\chi^*$ s.t. $\tau^*\chi^*$,

$\sigma'=Consequence(\gamma\theta_t)\chi^*\sigma$ satisfies: (2)

either for each ι,

if $\theta_\iota=\theta_t$

then $Result\pi_\iota\sigma'=Result\pi_\iota\sigma$

otherwise $\tau_\iota(Result\pi_\iota\sigma')$;

or $ErrorSet\sigma'$.

For if $AreT\theta^{\pi*}$

then $\gamma\theta_t=\theta_t$;

so (2) is trivially satisfied.

Otherwise let $\langle\pi_\iota,\zeta'\rangle=Choose\theta^{\pi*}\zeta$

and $\langle\chi,\theta'\rangle=\theta^{\pi*}\pi_\iota\sigma$

and $\theta'^{\pi*}=\theta^{\pi*}[\theta'/\pi_\iota]$

and $\gamma'=Combine\theta'^{\pi*}\zeta'$;

then $\gamma\theta_t\sigma=\langle\chi,\gamma'\theta_\iota\rangle$;

so $\sigma'=Consequence(\gamma\theta_t)\chi^*\sigma$

$=Consequence(\gamma'\theta_t)(\chi^*\dagger 1)(\chi_1\circ\chi\sigma)$;

but for $\pi\leftarrow(\pi^*\backslash\langle\pi_\iota\rangle)$,

$Result\pi(\chi_1\circ\chi\sigma) = Result\pi(\chi\sigma)$

since $\tau^*\chi^*$

$= Result\pi\sigma$

since $\chi\leftarrow Develop\theta_\iota\chi^*\sigma$;

Now if $\theta'=\theta_t$

then $\chi'^*=Develop\theta_\iota\chi^*\sigma=\langle\chi\rangle$;

so by (1),

either $ErrorSet(Consequence\theta_\iota\chi^*\sigma)$;

so $ErrorSet(\chi_1\circ\chi\sigma)$;

so $ErrorSet\sigma'$;

or $\tau_\iota(Result\pi_\iota(\chi\sigma))$;

hence $\tau_\iota(Result\pi_\iota(\chi_1\circ\chi\sigma))$;

also, θ' trivially satisfies (1) ;

and $\#Develop(\gamma'\theta_t)(\chi^*\dagger 1)(\chi_1\circ\chi\sigma)=\nu-1$;

so by inductive hypothesis,

either $ErrorSet\sigma'$

or for $\iota'\neq\iota$, if $\theta_{\iota'}=\theta_t$

then $Result\pi_{\iota'}\sigma'=Result\pi_{\iota'}(\chi_1\circ\chi\sigma)$;

otherwise $\tau_{\iota'}(Result\pi_{\iota'}\sigma')$;

and $Result\pi_\iota\sigma'=Result\pi_\iota(\chi_1\circ\chi\sigma)$;

hence for $\iota'\neq\iota$,

if $\theta_{\iota'}=\theta_t$

then $Result\pi_{\iota'}\sigma'=Result\pi_{\iota'}\sigma$

and $\tau_\iota(Result\pi_\iota\sigma')$;

which implies (2) since $\theta_\iota\neq\theta_t$.

If $\theta'\neq\theta_t$

then $\chi'^{*}=\langle\chi\rangle\hat{}\,Develop\theta'(\chi^{*}\dagger 1)(\chi_1\circ\chi\sigma)$;

so for any χ''^{*},σ_0, let $\chi_0=\lambda\sigma'.\sigma_0$;

then $Develop\theta'\chi''^{*}\sigma_0=Develop\theta_\iota(\langle\chi_0\rangle\hat{}\,\chi''^{*})\sigma\dagger 1$;

so condition (1) for θ' follows from condition for θ_ι ;

so again by inductive hypothesis applied to $\gamma'\theta_t$,

either for $\iota'\neq\iota$,

if $\theta_{\iota'}=\theta_t$

then $Result\pi_{\iota'},\sigma'=Result\pi_{\iota'},(\chi_1\circ\chi\sigma)$

otherwise $\tau_{\iota'},(Result\pi_{\iota'},\sigma')$

and $\tau_\iota(Result\pi_\iota\sigma')$;

hence since $\theta_\iota\neq\theta_t$

if $\theta_{\iota'}=\theta_t$

then $Result\pi_{\iota'},\sigma'=Result\pi_{\iota'},\sigma$

otherwise $\tau_{\iota'},(Result\pi_{\iota'},\sigma')$;

or $ErrorSet\sigma'$;

Thus (1) ⇒ (2);

Now each γ_ι is Regular wrt τ,τ',τ_ι, and JumpFree;

so by 2.4.4,

$\forall\chi^{*},\sigma_0$ s.t. $\tau^{*}\chi^{*}$,

either $\exists\varepsilon_\iota$ s.t. $\tau_\iota\varepsilon_\iota$

and $\forall\kappa$, $Develop(\gamma_\iota\kappa)\chi^{*}\sigma_0=$

$Develop(\gamma_\iota\kappa_t)\chi^{*}\sigma_0\hat{}\,Develop(\kappa\varepsilon_\iota)(\chi^{*}\dagger\nu)\sigma_1$

or $Develop(\gamma_\iota\kappa)\chi^{*}\sigma_0$ is independent of κ

and $ErrorSet\sigma_1$;

ie either $\forall\theta$, $Develop(\gamma''_\iota\theta)\chi^{*}\sigma_0=$

$\chi'^{*}\hat{}\langle\chi'\rangle\hat{}\,Develop\theta(\chi^{*}\dagger(\nu+1))\sigma_2$

where $\chi'^{*}=Develop(\gamma_\iota\kappa_t)\chi^{*}\sigma_0$

and $\chi'=Signal\pi_\iota\varepsilon_\iota$

and $\sigma_2 = \chi' \sigma_1$;

so $\forall\theta$, $Develop(\gamma''_\iota\theta)\chi^*\sigma_0 = \chi''^* \hat{} Develop\theta(\chi^* \dagger \nu_1)\sigma_2$

where $\chi''^* = Develop(\gamma''_\iota\theta_t)\chi^*\sigma_0$

etc;

so, since $\tau'^*\chi'^*$

$Develop(\gamma''_\iota\theta_t)\chi^*\sigma_0 = \chi'^* \hat{} (\chi')$ satisfies (1);

and also $\#Develop(\gamma''_\iota\theta_t)\chi^*\sigma_0 > 0$;

so $\gamma''_\iota\theta_t \neq \theta_t$;

or $\forall\theta$, $Develop(\gamma''_\iota\theta)\chi^*\sigma_0 = Develop(\gamma_\iota(\gamma'_\iota\theta))\chi^*\sigma_0$

$= Develop(\gamma_\iota\theta_t)\chi^*\sigma_0$;

so $Develop(\gamma''_\iota\theta)\chi^*\sigma_0$ is independent of θ

and $\tau'^*(Develop(\gamma''_\iota\theta_t)\chi^*\sigma_0)$;

also $Consequence(\gamma''_\iota\theta_t)\chi^*\sigma_0$

$- Consequence(\gamma_\iota(\gamma'_\iota(\theta_t))\chi^*\sigma_0$

$= Consequence(\gamma_\iota\kappa_t)\chi^*\sigma_0$

$= \sigma_1$;

so $ErrorSet(Consequence(\gamma''_\iota\theta_t)\chi^*\sigma_0)$;

so again $(\gamma''_\iota\theta_t)$ satisfies (1);

and furthermore (unless $ErrorSet\sigma_0$) $\gamma''_\iota\theta_t \neq \theta_t$.

Thus for each ι, $\theta_\iota = \gamma''_\iota\theta_t$ satisfies (1) and $\theta_\iota \neq \theta_t$;

so for any σ, χ^* s.t. $\tau^*\chi^*$,

$\sigma' = Consequence(Run_p\gamma''^*\pi^*\theta_t)\chi^*\sigma$ satisfies

for each ι, $\tau_\iota(Result\pi_\iota\sigma')$

or $ErrorSet\sigma'$;

but from 2.4.7,

$\forall\theta$, $Develop(Run_p\gamma''^*\pi^*\theta)\chi^*\sigma = \chi''^* \hat{} Develop\theta(\chi^* \dagger \nu)\sigma'$

where $\chi''^{*}=Develop(Run_{p}\gamma''^{*}\pi^{*}\theta_{t})\chi^{*}\sigma$;

so $\forall\theta$, $Develop(\gamma\theta)\chi^{*}\sigma=\chi''^{*}\hat{}Develop(UnlessError\theta)(\chi^{*}\dagger\nu)\sigma'$;

but if $\sim ErrorSet\sigma'$

then for each ι, $\tau_{\iota}(Result\pi_{\iota}\sigma')$;

so $Develop(\gamma\theta)\chi^{*}\sigma=\chi''^{*}\hat{}\langle\chi\rangle\hat{}Develop\theta(\chi^{*}\dagger(\nu+1))\sigma_{1}$

where $\chi=CheckError\ False$

and $\sigma_{1}=\chi\sigma'=\sigma'$;

so $Develop(\gamma\theta)\chi^{*}\sigma=Develop(\gamma\theta_{t})\chi^{*}\sigma\hat{}Develop\theta(\chi^{*}\dagger\nu)\sigma'$

where ν and σ' satisfy the required conditions;

otherwise

$Develop(\gamma\theta)\chi^{*}\sigma = \chi''^{*}\hat{}\langle CheckError\ True\rangle$

$= Develop(\gamma\theta_{t})\chi^{*}\sigma$;

which is independent of θ ;

and $Consequence(\gamma\theta_{t})\chi^{*}\sigma = CheckError\ True\ \sigma' = \sigma'$;

so $ErrorSet(Consequence(\gamma\theta_{t})\chi^{*}\sigma)$.

QED

Theorem 2.4:

If ρ^{*},η satisfy

for any σ, if $\pi=\eta[\![\mathbf{proc}]\!]$ and $\pi^{*}=NewProcs\pi(\#\rho^{*})\sigma$

each $\rho_{\iota}\eta'$ is Regular wrt $(Invariant\pi^{*})$, $(Subvariant\pi^{*})$,

and some H-Monotonic $\tau_{\iota\eta'}$

where $\eta'=\eta[\pi_{\iota}/\mathbf{proc}]$;

then $Eval\rho^{*}\eta$ is Regular wrt $(Invariant\pi^{*})$, $(Subvariant\langle\pi\rangle)$

and $(\lambda\varepsilon^{*}.\forall\iota,\tau_{\iota\eta}\varepsilon_{\iota})$

where *Eval* may be $Eval_{3}$ or, if each $\rho_{\iota}\eta'$ is JumpFree, $Eval_{4}$.

Proof

let $\tau=(Invariant\pi^{*})$

and $\tau'=(Subvariant\pi^{*})$

and $\tau''=(Subvariant\langle\pi\rangle)$;

$\gamma_\iota=\rho_\iota\eta_\iota$ clearly satisfies the hypotheses of 2.4.8 and 2.4.9;

so $\gamma=Run\gamma''^*\pi^*\circ UnlessError$ satisfies the conclusion;

ie for any χ_0^*,σ_0 s.t. $\tau^*\chi_0^*$,

let $\pi^*=NewProcs\pi(\#\rho^*)\sigma_0$

and $\sigma_1=CheckProcs\pi\pi^*\sigma_0$

and $\chi_1^*=\chi_0^*\dagger 1$;

then either $\forall\theta$, $Develop(\gamma\theta)\chi_1^*\sigma_1=\chi'^*\hat{}Develop\theta\chi_2^*\sigma_2$

where $\chi'^*=Develop(\gamma\theta_t)\chi_1^*\sigma_1$

and $\chi_2^*=\chi_1^*\dagger(\#\chi'^*)$

and $\sigma_2=Consequence(\gamma\theta_t)\chi_1^*\sigma_1$;

and for each ι, $\tau_{\iota\eta_\iota}\varepsilon_\iota$

where $\varepsilon^*=Results\pi^*\sigma_2$;

hence by H-Monotonicity,

for each ι, $\tau_{\iota\eta}\varepsilon_\iota$;

so $Develop(Eval\rho^*\eta\kappa')\chi_0^*\sigma_0$ =

$\langle\chi_1\rangle\hat{}\chi'^*\hat{}\langle\chi_2\rangle\hat{}Develop(\kappa'\varepsilon^*)\chi_3^*\sigma_3$

where $\chi_1=CheckProcs\pi\pi^*$

and $\chi_2=CheckResults\pi^*\varepsilon^*$

and $\chi_3^*=\chi_2^*\dagger 1$

and $\sigma_3=\chi_2\sigma_2=\sigma_2$;

so $Develop(Eval\rho^*\eta\kappa')\chi_0^*\sigma_0$ =

$Develop(Eval\rho^*\eta\kappa_t')\chi_0^*\sigma_0\hat{}Develop(\kappa'\varepsilon^*)\chi_3^*\sigma_3$;

and for each ι, $\tau_{\eta}\varepsilon_\iota$;

furthermore, by 2.4.6 and 2.4.7,

$\gamma\theta_t$ is Regular wrt τ,τ' ;

so $\tau'^*\chi'^*$;

so $\tau''^*\chi'^*$;

and χ_1 and χ_2 certainly satisfy τ'' ;

or $Develop(\gamma\theta)\chi_1^*\sigma_1$ is independent of θ ;

so $\forall\kappa'$, $Develop(Eval\rho^*\eta\kappa')\chi_0^*\sigma_0 = \chi_1 \hat{} \chi'^*$

which is independent of κ' ;

and as above, $\tau''^*\chi'^*$;

and $\tau''\chi_1$;

Hence $Eval\rho^*\eta$ is Regular wrt $\tau, \tau'', (\varepsilon^*.\forall\iota, \tau_\iota\varepsilon_\iota)$.

QED

Corollary

For any η, if each $X_\iota[\![\Xi_\iota]\!]$ satisfies

for any η_ι s.t. $\eta[\![\mathbf{proc}]\!] \leq \eta_\iota[\![\mathbf{proc}]\!]$

$X_\iota[\![\Xi_\iota]\!]\eta_\iota$ is Regular wrt ($Invariant\langle \eta_\iota[\![\mathbf{proc}]\!]\rangle$),

($Subvariant\langle \eta_\iota[\![\mathbf{proc}]\!]\rangle$) and some H-Monotonic $\tau_{\iota\eta_\iota}$

then $XEval[\![\Xi^*]\!]\eta$ is Regular wrt ($Invariant\langle \eta[\![\mathbf{proc}]\!]\rangle$),

($Subvariant\langle \eta[\![\mathbf{proc}]\!]\rangle$) and $(\lambda\varepsilon^*.\forall\iota, \tau_{\iota\eta}\varepsilon_\iota)$.

Proof

Take any η and let $\pi = \eta[\![\mathbf{proc}]\!]$;

and take any σ ;

let $\pi^* = NewProcs\pi(\#\Xi^*)\sigma$

and $\eta_\iota = \eta[\pi_\iota/\mathbf{proc}]$;

then since $Invariant\pi^*\chi \Rightarrow Invariant\langle \pi_\iota\rangle\chi$

and $Subvariant\langle \pi_\iota\rangle\chi \Rightarrow Subvariant\pi^*\chi$,

each $X_\iota[\![\Xi_\iota]\!]\eta_\iota$ is Regular wrt ($Invariant\pi^*$), ($Subvariant\pi^*$),

and $\tau_{\iota\eta_\iota}$;

so $Eval(\mathop{\text{\Large\#}}_\iota X_\iota[\![\Xi_\iota]\!])\eta$ is Regular wrt ($Invariant\pi^*$),

($Subvariant\langle\pi\rangle$), $(\lambda\varepsilon^*.\forall\iota, \tau_{\iota\eta}\varepsilon_\iota)$

for $Eval = Eval_3$, or if each $X_\iota[\![\Xi_\iota]\!]$ is JumpFree, $Eval_4$;

and in particular for $Eval = \tau[\![\Xi^*]\!]\eta \rightarrow Eval_4, Eval_3$

where τ is a P-Test for the X_ι ;

ie $XEval[\![\Xi^*]\!]\eta$ is Regular wrt $(Invariant\langle \pi \rangle)$,

$(Subvariant\langle \pi \rangle)$, $(\lambda\varepsilon^*.\forall\iota, \tau_{\iota\eta}\varepsilon_\iota)$

since $Invariant\langle \pi \rangle \chi \Rightarrow Invariant\pi^*\chi$.

QED

2.5 The Primitive Operations

The last property needed of *Eval* concerns the operations which can be produced as a result of using it, with a view to the satisfaction of the Miscibility condition of theorem 1.4. This section will show that these operations are essentially those produced by the component commands, as would be expected, together with a certain restricted collection of process-manipulating operations; and thence that the operations produced by using *Eval* satisfy certain conditions, provided of course that the operations produced by the individual commands satisfy them also.

But first, the types of operation which will be encountered must be enumerated, by means of the following definition; and the subsequent lemma indicates how they relate to each other.

Definition

A state transformation is a *Permitted Operation* if it has one of the following forms:

1) $\lambda\sigma.\sigma$

2) *SetError*

3) *CheckError*τ

4) $Put\alpha\beta$

5) $CheckFetch\alpha\beta$

6) $CheckNew\pi\alpha$

7) $CheckFull\pi$

8) $CheckProcs\pi\pi^*$

9) $Signal\pi\varepsilon$

10) $CheckResults\pi^*\varepsilon^*$

Lemma 2.5.1:

If χ_1 and χ_2 are Permitted Operations then they commute except possibly in the following cases:

1) One is either *SetError* or $CheckFull\pi$;

2) One is $Put\alpha\beta_1$ and the other is $Put\alpha\beta_2$ or $CheckFetch\alpha\beta_2$;

3) One is $CheckNew\pi\alpha_1$ and the other is $CheckNew\pi\alpha_2$;

4) One is $CheckProcs\pi\pi_1^*$ and the other is $CheckProcs\pi\pi_2^*$;

5) One is $Signal\pi\varepsilon_1$ and the other is either $Signal\pi\varepsilon_2$ or $CheckResults\pi^*\varepsilon^*$ with $\pi \lessdot \pi^*$.

Proof

$(\lambda\sigma.\sigma)$ commutes with anything,

so types (1), (2) and (7) can be disregarded.

The remainder are conveniently classified as being of one of the forms:

i) $\lambda\sigma.\psi\sigma$;

or ii) $\lambda\sigma.(\phi\sigma \rightarrow \psi\sigma, \underline{?})$.

If one (say χ_1) is of type (i) and the other of type (ii), then $\chi_1 \circ \chi_2 = \lambda\sigma.\psi_1(\phi_2\sigma \rightarrow \psi_2\sigma, \underline{?})$

$= \lambda\sigma.\phi_2\sigma \rightarrow \psi_1\psi_2\sigma, \underline{?}$;

while $\chi_2 \circ \chi_1 = \lambda\sigma.\phi_2\psi_1\sigma \rightarrow \psi_2\psi_1\sigma, \underline{?}$;

so χ_1 and χ_2 commute if

$$\psi_1 \circ \psi_2 = \psi_2 \circ \psi_1 \text{ and } \phi_2 \circ \psi_1 = \phi_2 \ . \tag{1}$$

If both are of type (ii),

then $\chi_1 \circ \chi_2 = \lambda\sigma.(\phi_1(\phi_2\sigma \to \psi_2\sigma, \underline{?}) \to \psi_1(\phi_2\sigma \to \psi_2\sigma, \underline{?}), \underline{?})$

$= \lambda\sigma.((\phi_2\sigma \wedge \phi_1\psi_2\sigma) \to \psi_1\psi_2\sigma, \underline{?})$;

so χ_1 and χ_2 commute if

$$\phi_1 \circ \psi_2 = \phi_1,\ \phi_2 \circ \psi_1 = \phi_2 \text{ and } \psi_1 \circ \psi_2 = \psi_2 \circ \psi_1 \ . \tag{2}$$

If both are of type (i),

then each is either $Put\alpha_1\beta_1$ or $Signal\pi_1\varepsilon_1$;

which only fail to commute if

either both are $Put\alpha_1\beta_1$ and $\alpha_1 = \alpha_2$

or both are $Signal\pi_1\varepsilon_1$ and $\pi_1 = \pi_2$.

If χ_1 is of type (i) and χ_2 of type (ii) (or vice versa),

If $\chi_1 = Put\alpha_1\beta_1$

then $\psi_1 = \lambda\sigma.\sigma[\beta_1/\mathbf{store}/\alpha_1]$;

but for all possible forms for χ_2,

ψ_2 does not affect the ⟦**store**⟧ component,

hence $\psi_1 \circ \psi_2 = \psi_2 \circ \psi_1$;

and the only ϕ_2 which depends on the ⟦**store**⟧ component

occurs for $\chi_2 = CheckFetch\alpha_2\beta_2$;

and in this case,

$\phi_2 \circ \psi_1\sigma = (Fetch\alpha_2(\sigma[\beta_1/\mathbf{store}/\alpha_1]) = \beta_2)$

$= \phi_2\sigma$ if $\alpha_1 \neq \alpha_2$;

Similarly if $\chi_1 = Signal\pi_1\varepsilon_1$,

the only χ_2 which does not commute with χ_1 is

$CheckResults\pi^*\varepsilon^*$ where $\pi_1 \lessdot \pi^*$.

If both are of type (ii),
then the only cases in which (2) does not hold are when
(at least) one of ϕ_1, ψ_1, and (at least) on of ϕ_2, ψ_2
refer to the same component of the state,
so consider each component in turn:

⟦err⟧ :

each $\chi_\iota = \mathit{CheckError}\tau_\iota$;
so $\psi_\iota = \lambda\sigma.\sigma$;
so (2) trivially holds.

⟦store⟧ :

each $\chi_\iota = \mathit{CheckFetch}\alpha_\iota\beta_\iota$;
so again $\psi_\iota = \lambda\sigma.\sigma$ and (2) holds.

⟦area⟧ :

each $\chi_\iota = \mathit{CheckNew}\pi_\iota\alpha_\iota$ and assume $\pi_1 \neq \pi_2$;
for any σ,
if $\mathit{StoreFull}\pi_1\sigma$
then $\chi_1\sigma \stackrel{?}{=}$;
so $\chi_2(\chi_1\sigma) \stackrel{?}{=}$;
also $\chi_2\sigma \stackrel{?}{=}$ or $\chi_2\sigma = \sigma[\mathit{True}/\mathsf{use}/\alpha_2]$;
so $\chi_1(\chi_2\sigma) \stackrel{?}{=}$.

Similarly if $\mathit{StoreFull}\pi_2\sigma$.

Otherwise if $\mathit{New}\pi_\iota\sigma = \alpha_\iota$ for $\iota = 1,2$;
so that $\mathit{Owner}\alpha_\iota = \pi_\iota$, hence $\alpha_1 \neq \alpha_2$;
then $\chi_1\sigma = \sigma[\mathit{True}/\mathsf{area}/\alpha_1]$;

and by properties of *StoreFull* and *New*,

$\sim StoreFull\pi_2(\sigma[True/area/\alpha_1])$

and $New\pi_2(\sigma[True/area/\alpha_1]) = New\pi_2\sigma = \alpha_2$;

so $\chi_2(\chi_1\sigma) = \sigma[\langle True, True\rangle / area/\langle \alpha_1, \alpha_2\rangle]$;

otherwise $\chi_1(\chi_2\sigma) = \chi_2(\chi_1\sigma) = \underline{?}$.

Hence χ_1 and χ_2 commute if $\pi_1 \neq \pi_2$.

⟦use⟧ :

each $\chi_\iota = CheckProcs\pi_\iota\pi_\iota^*$;

and a similar argument shows that χ_1 and χ_2 commute so long as $\pi_\iota \neq \pi_2$.

⟦res⟧ :

each $\chi_\iota = CheckResults\pi_\iota^*\varepsilon_\iota^*$;

so again $\psi_\iota = \lambda\sigma.\sigma$ and (2) trivially holds.

QED

Now for the results about *Eval*. As previously, two lemmas are first needed about the two versions of *Run*, and then these are combined to give a result about either form of *Eval*.

Lemma 2.5.2:

If $\gamma^*, \pi^*, \chi^*, \sigma$ satisfy

each γ_ι is Regular wrt $(Invariant\langle \pi_\iota \rangle)$ and some τ_ι

and $(Invariant\pi^*)^*\chi^*$

and if $\chi \triangleleft Develop(Run_s\gamma^*\pi^*\theta_t)\chi^*\sigma$

then for some ι,

$\chi \triangleleft Develop(\gamma_\iota\theta_t)\chi_\iota^*\sigma_\iota$

for some $\chi_\iota^*, \sigma_\iota$ s.t. $(Invariant\langle \pi_\iota \rangle)^*\chi_\iota^*$.

Proof

let $\psi = Fix(\lambda\psi.\lambda\iota\theta.\iota \leq \#\gamma^* \rightarrow \gamma_\iota(\psi(\iota+1)\theta),\theta)$;

then it suffices to show (by induction on $\#\gamma^* - \iota$) that

if $\chi \lessdot Develop(\psi\iota\theta_t)\chi^*\sigma$

then $\chi \lessdot Develop(\gamma_\iota,\theta_t)\chi^*_\iota,\sigma_\iota$, for some $\iota' \geq \iota$.

If $\iota > \#\gamma^*$ then $\psi\iota\theta_t = \theta_t$ so no such χ exists;

and result is vacuously true.

If $\iota \leq \#\gamma^*$ then $\psi\iota\theta_t = \gamma_\iota(\psi(\iota+1)\theta_t)$;

so by Regularity of γ_ι

and since $(Invariant\pi^*)^*\chi^*$, so $(Invariant(\pi_\iota))^*\chi^*$,

either $Develop(\psi\iota\theta_t)\chi^*\sigma = Develop(\gamma_\iota(\psi(\iota+1)\theta_t)\chi^*\sigma$

$= Develop(\gamma_\iota\theta_t)\chi^*\sigma$;

so $\chi \lessdot Develop(\psi\iota\theta_t)\chi^*\sigma \Rightarrow \chi \lessdot Develop(\gamma_\iota\theta_t)\chi^*\sigma$;

or $Develop(\psi\iota\theta_t)\chi^*\sigma = Develop(\gamma_\iota\theta_t)\chi^*\sigma \hat{} Develop(\psi(\iota+1)\theta_t)\chi'^*\sigma'$

for some χ'^* satisfying $(Invariant\pi^*)^*\chi'^*$;

but by inductive hypothesis,

$\chi \lessdot Develop(\psi(\iota+1)\theta_t)\chi'^*\sigma' \Rightarrow$

$\chi \lessdot Develop(\gamma_\iota,\theta_t)\chi^*_\iota,\sigma_\iota$, for some $\iota' \geq \iota+1$;

hence $\chi \lessdot Develop(\psi\iota\theta_t)\chi^*\sigma \Rightarrow$

$\chi \lessdot Develop(\gamma_\iota,\theta_t)\chi^*_\iota,\sigma_\iota$, for some $\iota' \geq \iota$;

Putting $\iota = 1$ now gives the required result.

QED

Lemma 2.5.3:

If $\gamma^*,\pi^*,\chi^*,\sigma_0$ satisfy

each γ_ι is Regular wrt $(Invariant\langle \pi_\iota \rangle),\tau_\iota$

where $\tau_\iota=\lambda\chi.((\forall\pi\sigma,Result\pi(\chi\sigma)\neq Result\pi\sigma \Rightarrow \pi_\iota\leq\pi) \wedge$
$(\forall\sigma,ErrorSet\sigma \Rightarrow ErrorSet(\chi\sigma)) \wedge$
$(\forall\alpha\sigma,\sigma[\![use]\!]\alpha \Rightarrow (\chi\sigma)[\![use]\!]\alpha))$

and $(Invariant\pi^*)^*\chi^*$

and if $\chi\leftarrow Develop(Run_p\gamma^*\pi^*\theta_t)\chi^*\sigma_0$

then for some ι,

$\chi\leftarrow Develop(\gamma_\iota\theta_t)\chi^*_\iota\sigma_\iota$

for some $\chi^*_\iota,\sigma_\iota$, s.t. $(Invariant\langle \pi_\iota \rangle)^*\chi^*_\iota$.

Proof

let $\theta^{\pi*}=\langle \pi^*,\lambda\pi_\iota.\theta_\iota\rangle$

where $\theta_\iota=\gamma_\iota\theta_t$

and $\zeta=NewSchedule\pi^*$;

Now show by induction on $\#Develop(Combine\theta^{\pi*}\zeta\theta_t)\chi^*\sigma_0$,

that if $\chi\leftarrow Develop(Combine\theta^{\pi*}\zeta\theta_t)\chi^*\sigma_0$

then for some ι, $\chi\leftarrow Develop\theta_\iota\chi^*_\iota(\chi'_\iota\sigma_0)$ (1)

where $Invariant\langle \pi_\iota \rangle\chi'_\iota$.

If $AreT\theta^{\pi*}$ then $Combine\theta^{\pi*}\zeta\theta_t=\theta_t$;

so no such χ exists.

Otherwise let $\langle \pi_\iota,,\zeta'\rangle=Choose\theta^{\pi*}\zeta$

and $\langle \chi',\theta'\rangle=\theta^{\pi*}\pi_\iota,\sigma_0$;

then $Develop(Combine\theta^{\pi*}\zeta\theta_t)\chi^*\sigma_0$

$=\langle \chi'\rangle \hat{}\, Develop(Combine\theta'^{\pi*}\zeta'\theta_t)(\chi^*\dagger 1)(\chi_1\circ\chi'\sigma_0)$

where $\theta'^{\pi*}=\theta^{\pi*}[\theta'/\pi_\iota,]$;

so either $\chi=\chi'$ in which case $\chi\leftarrow Develop\theta_\iota,\chi^*\sigma_0$;

or $\chi\leftarrow Develop(Combine\theta'^{\pi*}\zeta'\theta_t)(\chi^*\dagger 1)(\chi_1\circ\chi'\sigma_0)$;

so by inductive hypothesis,

$\chi \lessdot \mathit{Develop}\theta'_\iota \chi^*_\iota(\chi'_\iota \circ \chi_1 \circ \chi' \sigma_0)$ for some ι (2)

where $\theta'_\iota = (\iota = \iota' \to \theta', \theta_\iota)$;

but if $\iota = \iota'$

then $(\chi') \hat{} \mathit{Develop}\theta'_\iota \chi^*_\iota(\chi'_\iota \circ \chi_1 \circ \chi' \sigma_0)$

$= \mathit{Develop}\theta_\iota((\chi'_\iota \circ \chi_1) \hat{} \chi^*_\iota)\sigma_0$

where $\mathit{Invariant}(\pi_\iota)\chi'_\iota$

and $\mathit{Invariant}\pi^*\chi_1$ so $\mathit{Invariant}(\pi_\iota)\chi_1$;

so $\mathit{Invariant}(\pi_\iota)(\chi'_\iota \circ \chi_1)$;

while if $\iota \neq \iota'$

then $\mathit{Invariant}(\pi_\iota)\chi'_\iota$ and $\mathit{Invariant}(\pi_\iota)\chi_1$;

and by Regularity, τ_ι, χ' ;

so $\mathit{Invariant}(\pi_\iota)\chi'$;

so $\mathit{Invariant}(\pi_\iota)(\chi'_\iota \circ \chi_1 \circ \chi')$;

so (2) gives required form.

Hence (1) holds;

ie $\chi \lessdot \mathit{Develop}\theta_\iota \chi^*_\iota \sigma_\iota$ for some $\iota, \chi^*_\iota, \sigma_\iota$.

QED

Lemma 2.5.4:

If $\rho^*, \eta, \chi^*, \sigma_0$ satisfy

$\forall \eta'$, each $\rho_\iota \eta'$ is Regular wrt $(\mathit{Invariant}(\eta'[\![\mathbf{proc}]\!]))$,

$(\mathit{Subvariant}(\eta'[\![\mathbf{proc}]\!]))$ and some τ'_ι

and $(\mathit{Invariant}(\eta[\![\mathbf{proc}]\!]))^*\chi^*$;

and if $\chi \lessdot \mathit{Develop}(\mathit{Eval}\rho^*\eta\kappa'_t)\chi^*\sigma_0$

where Eval may be Eval_3

or, if $\rho_\iota \eta$ is JumpFree for all η, Eval_4 ;

then either $\chi \lessdot \mathit{Develop}(\rho_\iota \eta[\pi_\iota/\mathbf{proc}]\kappa_t)\chi^*_\iota \sigma_\iota$

for some $\iota, \chi^*_\iota, \sigma_\iota$ s.t. $(\mathit{Invariant}(\pi_\iota))^*\chi^*_\iota$

where $\pi^* = NewProcs\pi(\#\rho^*)\sigma_0$;

or χ has one of the following forms:

i) $CheckProcs\pi\pi^*$ where $\pi = \eta[\![\,\mathbf{proc}\,]\!]$ and $\forall \pi_1 \lessdot \pi^*, \pi < \pi_1$;

ii) $Signal\pi_1\varepsilon$ where $\eta[\![\,\mathbf{proc}\,]\!] < \pi_1$;

iii) $CheckResults\pi^*\varepsilon^*$ where $\forall \pi_1 \lessdot \pi^*, \eta[\![\,\mathbf{proc}\,]\!] < \pi_1$;

iv) $CheckError\tau$.

Proof

let $\pi = \eta[\![\,\mathbf{proc}\,]\!]$

and $\pi^* = NewProcs\pi(\#\rho^*)\sigma_0$ (so $\forall \pi_1 \lessdot \pi^*, \pi \leq \pi_1$)

and $\eta_1 = \eta[\pi_1/\mathbf{proc}]$

and $\gamma^* = \mathop{\text{\Large\#}}_{1}^{\#\rho^*} \lambda\theta.\rho_1\eta_1(\lambda\varepsilon\sigma.\langle Signal\pi_1\varepsilon, \theta\rangle)$

and τ_1 be as in 2.5.3;

then $Develop(Eval\rho^*\eta\kappa_t')\chi^*\sigma_0$

$= \langle CheckProcs\pi\pi^*\rangle \hat{}\, Develop(Run\gamma^*\pi^*\theta_0)(\chi^* \dagger 1)\sigma_1$

where $\theta_0 = UnlessError(\lambda\sigma.\langle CheckResults\pi^*\varepsilon^*, \theta_t\rangle)$

where $\varepsilon^* = Results\pi^*\sigma$

and $\sigma_1 = \chi_1(CheckProcs\pi\pi^*\sigma_0)$;

hence either $\chi = (CheckProcs\pi\pi^*)$ which satisfies (i)

or $\chi \lessdot Develop(Run\gamma^*\pi^*\theta_0)(\chi^* \dagger 1)\sigma_1$;

but $\rho_1\eta_1$ is Regular wrt $(Invariant\langle \pi_1\rangle)$,

$(Subvariant\langle \pi_1\rangle)$, τ_1'

and therefore wrt $(Invariant\langle \pi_1\rangle)$, τ_1, τ_1' ;

while $\lambda\theta.\lambda\varepsilon\sigma.\langle Signal\pi_1\varepsilon, \theta\rangle$ is certainly Regular

wrt $(Invariant\langle \pi_1\rangle)$, τ_1, τ_1' ;

so γ_1 is Regular wrt $(Invariant\langle \pi_1\rangle)$, τ_1

and therefore wrt $\langle Invariant\langle \pi\rangle)$, $(Subvariant\langle \pi\rangle)$;

so by 2.4.6 and 2.4.7, $Run\gamma^*\pi^*$ is Regular wrt

$(Invariant\langle \pi\rangle)$, $(Subvariant\langle \pi\rangle)$;

so either $Develop(Run\gamma^*\pi^*\theta_0)(\chi^* \dagger 1)\sigma_1$

$= Develop(Run\gamma^*\pi^*\theta_t)(\chi^*\dagger 1)\sigma_1 \hat{} Develop\theta_0\chi'^*\sigma_2$

for some χ'^*, σ_2 ;

or $Develop(Run\gamma^*\pi^*\theta_0)(\chi^*\dagger 1)\sigma_1$

$= Develop(Run\gamma^*\pi^*\theta_t)(\chi^*\dagger 1)\sigma_1$;

in either case,

either $\chi \leftarrow Develop(Run\gamma^*\pi^*\theta_t)(\chi^*\dagger 1)\sigma_1$ (1)

or $\chi \leftarrow Develop\theta_0\chi'^*\sigma_2$; (2)

but in case (1), by 2.5.2 and 2.5.3,

$\chi \leftarrow Develop(\gamma_\iota\theta_t)\chi_\iota^*\sigma_\iota$

for some $\iota, \chi_\iota^*, \sigma_\iota$ s.t. $(Invariant\langle \pi_\iota \rangle)^*\chi_\iota^*$

$= Develop(\rho_\iota\eta_\iota(\lambda\varepsilon\sigma.\langle Signal\pi_\iota\varepsilon, \theta_t \rangle))\chi_\iota^*\sigma_\iota$;

but $\rho_\iota\eta_\iota$ is Regular,

so either $Develop(\gamma_\iota\theta_t)\chi_\iota^*\sigma_\iota$

$= Develop(\rho_\iota\eta_\iota\kappa_t)\chi_\iota^*\sigma_\iota \hat{} \langle Signal\pi_\iota\varepsilon_\iota \rangle$

for some ε_ι s.t. $\tau_\iota'\varepsilon_\iota$;

or $Develop(\gamma_\iota\theta_t)\chi_\iota^*\sigma_\iota = Develop(\rho_\iota\eta_\iota\kappa_t)\chi_\iota^*\sigma_\iota$;

in either case,

either $\chi \leftarrow Develop(\rho_\iota\eta_\iota\kappa_t)\chi_\iota^*\sigma_\iota$

or $\chi = (Signal\pi_\iota\varepsilon_\iota)$ which satisfies (ii);

while in case (2),

if $ErrorSet\sigma_2$

then $Develop\theta_0\chi'^*\sigma_2 = \langle CheckError\ True \rangle$;

and if $\sim ErrorSet\sigma_2$

then $Develop\theta_0\chi'^*\sigma_2$

$= \langle CheckError\ False \rangle \hat{} \langle CheckResults\pi^*\varepsilon^* \rangle$

where $\varepsilon^* = Results\pi^*(\chi_1'\sigma_2)$;

so $\chi = (CheckError\tau)$ which satisfies (iv);

or $\chi = (CheckResults\pi^*\varepsilon^*)$ satisfying (iii);

=

QED

The previous result can be tidied up by introducing the complex and rather more stringent condition which the operations produced by the semantic equations will be required to satisfy, even though this condition depends on parts of the environment which have different structures for the different languages considered in chapters 3 to 5. Because of this, the condition will be defined in terms of the predicate $Denoted\epsilon[D\rightarrow Ide\rightarrow H\rightarrow T]$ which checks whether the identifier denotes the given value in the given environment. The only significant property of this function from the point of view of theorem 2.5 is that it doesn't depend on the $[\![\mathbf{proc}]\!]$ component of the environment.

Definition

A primitive state transformation χ is *Sanctioned* by η, I_1^*, I_2^* if it is a permitted operation and

1) if χ is $Put\alpha\beta$ then
   either $Denoted\alpha I\eta$ for some $I \triangleleft I_1^*$
   or $\eta[\![\mathbf{proc}]\!] \leq Owner\alpha$;
2) if χ is $CheckFetch\alpha\beta$ then
   either $Denoted\alpha I\eta$ for some $I \triangleleft I_2^*$
   or $\eta[\![\mathbf{proc}]\!] \leq Owner\alpha$;
3) if χ is $CheckNew\pi\alpha$ then
   $\eta[\![\mathbf{proc}]\!] \leq \pi = Owner\alpha$;
4) if χ is $CheckFull\pi$ then
   $\eta[\![\mathbf{proc}]\!] \leq \pi$;
5) if χ is $Signal\pi\varepsilon$ then
   $\eta[\![\mathbf{proc}]\!] \leq \pi$;
6) if χ is $CheckResults\pi^*\varepsilon^*$ then

$\forall \pi_1 \prec \pi^*, \eta[\![\text{proc}]\!] < \pi_1$;

7) if χ is *CheckProcs*$\pi\pi^*$ then

$\eta[\![\text{proc}]\!] \leq \pi$ and $\forall \pi_1 \prec \pi^*, \eta[\![\text{proc}]\!] < \pi_1$;

Theorem 2.5:

For any η, if ρ^* satisfies

$\forall \pi_1, \chi_1^*, \sigma_1$ s.t. (*Invariant*(π_1))$^*\chi_1^*$,

each component of *Develop*$(\rho_1 \eta_1 \kappa_t)\chi_1^*\sigma_1$

is Sanctioned by η_1, I_1^*, I_2^*

where $\eta_1 = \eta[\pi_1/\text{proc}]$;

and $\forall \eta'$, each $\rho_1 \eta'$ is Regular wrt (*Invariant*$(\eta'[\![\text{proc}]\!])$),

(*Subvariant*$(\eta'[\![\text{proc}]\!])$) and some τ_1' ;

then $\forall \eta, \chi^*, \sigma$ s.t. (*Invariant*$(\eta[\![\text{proc}]\!])$)$^*\chi^*$,

each component of *Develop*(*Eval*$\rho^* \eta \kappa_t')\chi^*\sigma_0$

is Sanctioned by η, I_1^*, I_2^*.

Proof

let $\pi = \eta[\![\text{proc}]\!]$

and $\pi^* =$ *NewProcs*$\pi(\#\rho^*)\sigma_0$

and $\eta_1 = \eta[\pi_1/\text{proc}]$

and let $\chi \prec$ *Develop*(*Eval*$\rho^* \eta \kappa_t')\chi^*\sigma_0$;

then by 2.5.4,

either for some $\iota, \chi_1^*, \sigma_1$ s.t. (*Invariant*(π_1))$^*\chi_1^*$

$\chi \prec$ *Develop*$(\rho_1 \eta_1 \kappa_t)\chi_1^*\sigma_1$;

hence by hypothesis, χ is Sanctioned by η_1, I_1^*, I_2^* ;

but since $\pi < \pi_1$, this implies that

χ is Sanctioned by η, I_1^*, I_2^* ;

or χ has one the forms (i) to (iv) of 2.5.4,

all of which are Sanctioned by η and any I_1^*, I_2^* .

QED

CHAPTER 3: A VERY SIMPLE LANGUAGE

"'I only took the regular course... Reeling and Writhing, of course, to begin with, and then the different branches of Arithmetic - Ambition, Distraction, Uglification and Derision."

Alice's Adventures in Wonderland: Chap 9.

In this chapter, the first of three example languages is described, and syntactic (ie compiler-checkable) conditions are found on the use of parallelism. The language will be known as ELM (Elementary Language Model) and it will have only very basic facilities - definitions of simple variables, assignments, simple loops, tests and grouping of commands into blocks.

The proofs of many of the theorems in this and the next two chapters are long, cumbersome and repetitive, as they involve consideration of many very similar cases. Parts of these will therefore be given in outline only, where they are similar to other parts which are given in detail.

3.0 Description of ELM

The syntax of ELM requires six syntactic domains: a domain **Bas** of objects representing basic constants (eg numerals); a domain **Ide** of identifiers; a domain **Ope** of (diadic) operators (+ = etc). It will be assumed without further discussion that elements of these domains can be recognised and distinguished, and also that there exist functions $B \epsilon [\mathbf{Bas} \rightarrow \mathbf{B}]$ and $W \epsilon [\mathbf{Ope} \rightarrow \mathbf{V}^* \rightarrow \mathbf{V}]$ which define the meanings of constants and operators. The

remaining three domains are recursively defined in terms of each other. The domain **Com** of commands of the language is a sum of six types representing: i) assignment, ii) grouping of a list of commands into a block (which is labelled with some identifier), iii) an "escape" command which jumps to the end of the enclosing block with the appropriate identifier, iv) the "while" loop, v) conditional command, and vi) association of a declaration with a command. Expressions, of the domain **Exp**, can also be of six types: i) a basic constant, ii) an identifier, iii) two subexpressions connected by an operator, iv) a subexpression together with a command to be executed before it, v) a conditional expression, and vi) association of a declaration with an expression. The last domain **Dec** of declarations has only three forms: i) declaration of an identifier to denote a constant value which cannot thereafter be changed by assignment, ii) declaration of an identifier to denote a new location whose contents are initialised to a specified value, and iii) combination of these in simultaneous declarations. Constructions such as bracketted expressions and monadic operators could easily be included, but they add nothing new and only encumber the semantic equations and proofs. In common with [23], details of syntax analysis - unambiguous parsing, precedence of operators, etc. - are disregarded.

The form of jumps available by the labelled block/**escape** mechanism is obviously more restricted than the traditional label/**goto** mechanism, but together with suitable loop constructions, it seems to be quite adequate for most purposes, and it simplifies many of the semantic equations, at the same time demonstrating many of the techniques required to handle

full jumps, and indeed probably clarifying some of the issues involved.

A few new semantic domains are also needed. First, a domain B of basic values, eg numbers, characters, truth values (as distinct from their representations in a program). The other new domain is D, which contains all those values which identifiers are allowed to denote; like E and V, it is a sum of other value domains, in this case L+B. Further details must also be given of domains which were incompletely specified in the last chapter. Thus E=L+B and V=B, indicating that "pointer" variables are not permitted, as L is not a summand of V. For the environment H, apart from the components described in chapter 2, one will be needed to indicate the current denotation of each identifier as a "variable", hence:

$$H = \mathbf{proc{:}P} \times \mathbf{var{:}[Ide \to D]} \times \mathbf{cont{:}[Ide \to C]}$$

The semantics of ELM can now be defined (see Appendix 1). The functions $C\epsilon[\mathbf{Com \to H \to C \to C}]$, $E\epsilon[\mathbf{Exp \to H \to K \to C}]$ and $D\epsilon[\mathbf{Dec \to H \to K' \to C}]$ are the principal semantic functions for the domains **Com**, **Exp** and **Dec** respectively; L and $R\epsilon[\mathbf{Exp \to H \to K \to C}]$ are used for evaluating expressions in L-mode and R-mode using the functions Lv and Rv respectively; (Rv may seem unduly complicated, since the final case can never occur in ELM, but the more general form will be needed in the next chapter;) $I\epsilon[\mathbf{Dec \to Ide^*}]$ gives a list of the identifiers defined in a given declaration. Note the use of *CheckFetch*, *CheckNew* and *CheckFull* to ensure Consistency. The reason for the use of *Break* in the equation for the **while** loop is to allow induction to be used in proofs, as will become apparent in due course. Four different versions of *Eval* are

used: *CEval*, *REval* and *DEval* apply *C*, *R* and *D* respectively to all their arguments, and *LREval* always has a pair of expressions as arguments, and applies *L* to the first and *R* to the second. The purpose of *ClaimProcs* in the two ⟦ **let** Δ **in** ...⟧ clauses will be clear in 3.5 - it concerns only the "Ownership" of newly claimed locations.

3.1 Consistency of ELM

It will now be shown that the semantic equations for ELM always produce commands which are Consistent. The proof, like most of the remaining theorems, will make use of structural induction on the syntactic categories, by which, in proving a result for a certain syntactic object, it is assumed inductively to be true for all strict subexpressions. Since **Com**, **Exp** and **Dec** are mutually recursive, it is necessary to prove the results for *C*, *E*, *L*, *R* and *D* simultaneously. In fact, the situation is usually rather more complicated, because in one case, (the **while** loop,) the induction would require the inductive hypothesis to be applied not to a strict subexpression but to the whole expression. Thus a double induction is required - first on some integer measure ν (which in this case is the parameter of Consistency, and later will be the length of a "Development"), and then on the syntactic structure. Thus, on the assumption that the result is true for measure less than ν, the structural induction shows it to be true for all cases of measure ν; this inductive step then shows the result is true in all cases. In practice, this means that in proving the case of a syntactic element Ξ and measure ν_0, the inductive hypothesis will assert

that the result is true for all cases of measure $<\nu_0$ and for all cases of proper components of Ξ and measure ν_0.

First, some preliminary lemmas are needed to show the Consistency of the subsidiary functions. The main theorem then follows.

Lemma 3.1.1:

For any $\nu>0$,

If θ is $(\nu-1)$-Consistent

then $\theta'=Update\alpha\beta\theta$ is ν-Consistent;

Hence if θ is Consistent, so is θ'.

Proof

$\theta'\neq\theta_t$, so take any σ_1, σ_2,

a) $\theta'\sigma_1\downarrow 2=\langle Put\alpha\beta,\theta\rangle\downarrow 2=\theta$ which is $(\nu-1)$-Consistent;

b) $(\theta'\sigma_1\downarrow 1)\sigma_1=Put\alpha\beta\sigma_1\neq\underline{?}$ (provided $\sigma_1\neq\underline{?}$) ;

c) $\theta'\sigma_1=\langle Put\alpha\beta,\theta\rangle=\theta'\sigma_2$;

ie θ' is ν-Consistent.

QED

Lemma 3.1.2:

For any $\nu>0$,

If κ is $(\nu-1)$-Consistent

then $\theta'=Cont s\alpha\kappa$ is ν-Consistent;

Hence if κ is Consistent, so is θ'.

Proof

Take any σ_1, σ_2;

let $\varepsilon_1=Fetch\alpha\sigma_1$

and $\varepsilon_2=Fetch\alpha\sigma_2$;

a) $\theta'\sigma_1\downarrow 2=\kappa\varepsilon_1$ which is $(\nu-1)$-Consistent since κ is;

b) $(\theta'\sigma_1\downarrow 1)\sigma_1 = \mathit{CheckFetch}\alpha\varepsilon_1\sigma_1 = \sigma_1$;

c) if $\varepsilon_1 = \varepsilon_2$

then $\theta'\sigma_1 = \langle \mathit{CheckFetch}\alpha\varepsilon_1, \kappa\varepsilon_1\rangle$

$= \langle \mathit{CheckFetch}\alpha\varepsilon_2, \kappa\varepsilon_2\rangle$

$= \theta'\sigma_2$;

otherwise $(\theta'\sigma_2\downarrow 1)\sigma_1 = \mathit{CheckFetch}\alpha\varepsilon_2\sigma_1 = ?$;

ie θ' is ν-Consistent.

QED

Lemma 3.1.3:

For any $\nu > 0$,

If κ is ν-Consistent

then $\theta = \mathit{NewLoc}\pi\kappa$ is ν-Consistent;

Hence if κ is Consistent, so is θ.

Proof

Take any σ_1, σ_2;

If $\mathit{StoreFull}\pi\sigma_1$

then a) $(\theta\sigma_1\downarrow 2) = \theta_t$ which is ν-Consistent;

b) $(\theta\sigma_1\downarrow 1)\sigma_1 = \mathit{CheckFull}\pi\sigma_1 = \sigma_1$;

c) If $\mathit{StoreFull}\pi\sigma_2$

then $\theta\sigma_1 = \langle \mathit{CheckFull}\pi, \theta_t\rangle = \theta\sigma_2$;

otherwise $(\theta\sigma_2\downarrow 1)\sigma_1 = \mathit{CheckNew}\pi(\mathit{New}\pi\sigma_2)\sigma_1 = ?$;

otherwise, let $\alpha_1 = \mathit{New}\pi\sigma_1$;

a) $(\theta\sigma_1\downarrow 2) = \kappa\alpha_1$ which is ν-Consistent;

b) $(\theta\sigma_1\downarrow 1)\sigma_1 = \mathit{CheckNew}\pi\alpha_1\sigma_1 = \sigma_1$;

c) If $\mathit{StoreFull}\pi\sigma_2$

then $(\theta\sigma_2\downarrow 1)\sigma_1 = \mathit{CheckFull}\pi\sigma_1 = ?$;

otherwise let $\alpha_2 = \mathit{New}\pi\sigma_2$;

if $\alpha_1 = \alpha_2$

then $\theta\sigma_1 = \langle \mathit{CheckNew}\pi\alpha_1, \kappa\alpha_1\rangle$

$=\langle CheckNew\pi\alpha_2, \kappa\alpha_2\rangle$

$=\theta\sigma_2$;

otherwise $(\theta\sigma_2\downarrow 1)\sigma_1 = CheckNew\pi\alpha_2\sigma_1 = \stackrel{?}{=}$;

ie θ is ν-Consistent.

QED

Lemma 3.1.4:

For any $\nu>0$,

If κ is ν-Consistent

then so are $L\nu\kappa$ and $R\nu\kappa$;

Hence if κ is Consistent, so are $L\nu\kappa$ and $R\nu\kappa$.

Proof

Take any $\varepsilon\epsilon E$;

if $\varepsilon\epsilon L$ then $L\nu\kappa\varepsilon=\kappa\varepsilon$ which is ν-Consistent;

if not then $L\nu\kappa\varepsilon=Error$ which is ν-Consistent;

if $\varepsilon\epsilon L$ then $R\nu\kappa\varepsilon=Conts\varepsilon\kappa$ which is ν-Consistent by 3.1.2;

if $\varepsilon\epsilon V$ then $R\nu\kappa\varepsilon=\kappa\varepsilon$ which is ν-Consistent;

otherwise $R\nu\kappa\varepsilon=Error$ which is ν-Consistent.

QED

Theorem 3.1:

If $\theta_0, \kappa_0, \kappa_0', \eta$ are ν-Consistent

then $C[\![\Theta]\!]\eta\theta_0$, $E[\![E]\!]\eta\kappa_0$, $L[\![E]\!]\eta\kappa_0$, $R[\![E]\!]\eta\kappa_0$ and $D[\![\Delta]\!]\eta\kappa_0'$

are ν-Consistent;

Hence if θ_0 etc are Consistent, so are $C[\![\Theta]\!]\eta\theta_0$ etc.

Proof

by induction on ν and on the syntactic structure.

First observe that the results for *L* and *R* follow immediately from that for *E* and 3.1.4.

Case $[\![E_1 := E_2]\!]$:

let $\kappa' = (\lambda\langle \alpha,\beta\rangle .\mathit{Update}\alpha\beta\theta_0)$

which is ν-Consistent by 3.1.1;

but by structural induction,

$L[\![E_1]\!]$ and $R[\![E_2]\!]$ preserve ν-Consistency;

hence by 2.2,

$C[\![\Theta]\!]\eta\theta_0 = \mathit{LREval}\langle [\![E_1]\!], [\![E_2]\!]\rangle \eta\kappa'$ is ν-Consistent.

Case $[\![$ §I Θ^* §I $]\!]$:

since η and θ_0 are both ν-Consistent,

$\eta' = \eta[\theta_0/\mathbf{cont}]$ is also ν-Consistent;

but by structural inductive hypothesis,

each $C[\![\Theta_\iota]\!]$ preserves ν-Consistency;

hence by 2.2,

$C[\![\Theta]\!]\eta\theta_0 = \mathit{CEval}[\![\Theta^*]\!]\eta'\theta_0$ is ν-Consistent.

Case $[\![$ **while** E **do** Θ_1 $]\!]$:

The recursive nature of the semantic function in this case necessitates the use of induction on ν.

Let $\theta_1 = C[\![\Theta]\!]\eta\theta_0$;

then by inductive hypothesis, θ_1 is $(\nu-1)$-Consistent;

hence $\mathit{Break}\theta_1$ is ν-Consistent;

(The presence of *Break* thus allows the inductive step to be made; this is the only reason for including it;)

so by structural induction,

$\theta_2 = C[\![\Theta_1]\!]\eta(\mathit{Break}\theta_1)$,

$\kappa = \lambda\varepsilon.\varepsilon\rightarrow\theta_2,\theta_0$,

and $\theta_1 = R[\![E]\!]\eta\kappa$

are all ν-Consistent.

Case ⟦**if** E **do** Θ_1⟧ :

θ_0 is ν-Consistent;

so by structural induction,

so is $C⟦\Theta_1⟧\eta\theta_0$;

hence so is $\kappa_1=(\lambda\varepsilon.\varepsilon\rightarrow C⟦\Theta_1⟧\eta\theta_0,\theta_0)$;

and so is $C⟦\Theta⟧\eta\theta_0=R⟦E⟧\eta\kappa_1$.

Case ⟦**escape** I⟧ :

η is ν-Consistent, so η⟦**cont**⟧⟦I⟧ is.

Case ⟦**let** Δ **in** Θ_1⟧ :

Consistency of η depends only on its ⟦**cont**⟧ component;

so, since η is ν-Consistent,

so is $\eta[\varepsilon^*/\mathbf{var}/I⟦\Delta⟧]$ for any ε^* ;

also *ClaimProcs* preserves Consistency,

by an argument similar to 3.1.3;

so the result follows from structural induction.

Case ⟦B⟧ :

Case ⟦I⟧ :

result follows immediately from ν-Consistency of κ_0.

Case ⟦E_1 Ω E_2⟧ :

result follows from structural induction on $R⟦E_1⟧$ and $R⟦E_2⟧$

and from 2.2.

Case ⟦Θ **before** E_1⟧ :

Case ⟦$E_1 \rightarrow E_2, E_3$⟧ :

result follows immediately from structural induction.

Case ⟦ **let** Δ **in** E_1 ⟧ :

exactly as case ⟦ **let** Δ **in** Θ_1 ⟧.

Case ⟦ I ≡ E ⟧ :

immediate from structural induction.

Case ⟦ I = E ⟧ :

κ_0' is ν-Consistent, so by 3.1.1,

$\kappa_1 = (\lambda\alpha.\mathit{Update}\,\alpha\varepsilon(\kappa'\langle\alpha\rangle))$ is ν-Consistent for any ε ;

so by 3.1.3,

$\kappa_2 = (\lambda\varepsilon.\mathit{NewLoc}\,\pi\kappa_1)$ is ν-Consistent $\quad(\pi = \eta[\![\,\mathbf{proc}\,]\!])$;

so by structural induction,

$D[\![\,\Delta\,]\!]\eta\kappa_0' = R[\![\,E\,]\!]\eta\kappa_2$ is ν-Consistent.

Case ⟦ Δ_1 **and** Δ_2 ⟧ :

immediate from structural induction and 2.2.

QED

3.2 Termination in ELM

The T-List condition of a P-Test is a little more complicated than that of Consistency, since obviously not all sections of a program can be expected to be JumpFree. It is, however, a simple matter to search through a piece of program text listing all the jumps (ie **escape** commands) which are not "bound" by enclosing blocks with the same label. This is precisely what the function $\mathit{Jumps} \in [\mathbf{Com+Exp+Dec}] \to \mathbf{Ide}^*$ does, and it is straightforward, if laborious, to show that this function

has the desired effect.

Lemma 3.2.1:

If I^* is a T-List for $E[\![E]\!]$

then it is a T-List also for $L[\![E]\!]$ and $R[\![E]\!]$.

Proof

Take any η',χ^*,σ_0 s.t. $\eta'[\![\mathbf{cont}]\!]=\lambda I.\theta_t$

and assume that η always satisfies $\eta'=\eta[\lambda I.\theta_t/\mathbf{cont}]$;

let $\sigma_1=Consequence(E[\![E]\!]\eta'\kappa_t)\chi^*\sigma_0$

and $\chi_1^*=Develop(E[\![E]\!]\eta'\kappa_t)\chi^*\sigma_0$

and $\chi_2^*=Develop(R[\![E]\!]\eta'\kappa_t)\chi^*\sigma_0$

and $\chi'^*=\chi^*\dagger(\#\chi_1^*)$;

now since I^* is a T-List for $E[\![E]\!]$,

$\surd$ or ? or some $I\lhd I^*$ is a Terminator for $E[\![E]\!],\eta',\chi^*,\sigma_0$;

ie a) $\exists\varepsilon$, $\forall\eta,\kappa$,

$$Develop(R[\![E]\!]\eta\kappa)\chi^*\sigma_0 = Develop(E[\![E]\!]\eta(R\upsilon\kappa))\chi^*\sigma_0$$
$$= \chi_1^*\hat{}Develop(R\upsilon\kappa\varepsilon)\chi'^*\sigma_1 ;$$

if $\varepsilon\in L$

then $Develop(R\upsilon\kappa\varepsilon)\chi'^*\sigma_1$
$= \langle CheckFetch\varepsilon\beta\rangle\hat{}Develop(\kappa\beta)(\chi'^*\dagger 1)\sigma_2$

where $\beta=Fetch\varepsilon\sigma_1$

and $\sigma_2=CheckFetch\varepsilon\beta\sigma_1=\sigma_1$;

so $\chi_2^*=\chi_1^*\hat{}\langle CheckFetch\varepsilon\beta\rangle$;

so $\forall\eta,\kappa$, $Develop(E[\![E]\!]\eta\kappa)\chi^*\sigma_0=\chi_2^*\hat{}Develop(\kappa\beta)\chi''^*\sigma_2$

where $\chi''^*=\chi^*\dagger(\#\chi_1^*+1)=\chi^*\dagger(\#\chi_2^*)$;

hence $\surd$ is a Terminator for $R[\![E]\!],\eta',\chi^*,\sigma_0$;

if $\varepsilon\in V$

then $Develop(R\upsilon\kappa\varepsilon)\chi'^*\sigma_1 = Develop(\kappa\varepsilon)\chi'^*\sigma_1$;

so $\chi_2^*=\chi_1^*$

and $Develop(R[\![E]\!]\eta\kappa)\chi^*\sigma_0 = \chi_2^* \hat{} Develop(\kappa\varepsilon)\chi'^*\sigma_1$;

hence again, ✓ is a Terminator for $R[\![E]\!]$, etc.

if $\varepsilon \notin L+V$

then $Develop(R\nu\kappa\varepsilon)\chi'^*\sigma_1 = \langle SetError \rangle$;

so $\chi_2^* = \chi_1^* \hat{} \langle SetError \rangle$;

so $\forall \eta, \kappa$, $Develop(R[\![E]\!]\eta\kappa)\chi^*\sigma_0 = \chi_2^*$

and $ErrorSet(Consequence(R[\![E]\!]\eta'\kappa_t)\chi^*\sigma_0)$;

ie ? is a Terminator for $R[\![E]\!]$, etc.

or b) ? or some $I \lhd I^*$ is a Terminator for $E[\![E]\!], \eta', \chi^*, \sigma_0$;

in which case it immediately follows that

$R[\![E]\!], \eta', \chi^*, \sigma_0$ have the same Terminator.

Thus, in either case, I^* is a Terminator for $R[\![E]\!], \eta', \chi^*, \sigma_0$.

The proof for $L[\![E]\!]$ is similar except that the conditions on ε in case (a) are rearranged.

QED

Theorem 3.2:

For any $\Theta \in \mathbf{Com}$, $E \in \mathbf{Exp}$, $\Delta \in \mathbf{Dec}$,

$Jumps[\![\Theta]\!]$ is a T-List for $C[\![\Theta]\!]$;

$Jumps[\![E]\!]$ is a T-List for $E[\![E]\!]$, $L[\![E]\!]$ and $R[\![E]\!]$;

$Jumps[\![\Delta]\!]$ is a T-List for $D[\![\Delta]\!]$.

Proof

The results for $L[\![E]\!]$ and $R[\![E]\!]$ follow from that for $E[\![E]\!]$ and from 3.2.1.

For the others, take any η', χ^*, σ_0 s.t. $\eta'[\![\mathbf{cont}]\!] = \lambda I.\theta_t$ and assume that η always satisfies $\eta' = \eta[\lambda I.\theta_t / \mathbf{cont}]$;

Then use induction on $\nu = \#(Develop(C[\![\Theta]\!]\eta'\theta_t)\chi^*\sigma_0)$ etc. and on

the syntactic structure.

Case $[\![E_1 := E_2]\!]$:

let $I_1^* = Jumps[\![E_1]\!]$ and $I_2^* = Jumps[\![E_2]\!]$;

then from inductive hypothesis,

$I^* = I_1^* \hat{} I_2^*$ is a T-List for $L[\![E_1]\!]$ and for $R[\![E_2]\!]$;

so by 2.3, I^* is a T-List for $LREval\langle [\![E_1]\!], [\![E_2]\!] \rangle$;

the remainder of the proof proceeds as in 3.2.1.

Case $[\![\S I\ \theta^*\ \S I]\!]$:

For each ι,

let $I_\iota^* = Jumps[\![\theta_\iota]\!]$;

and let $I^* = \bigwedge_{\iota=1}^{\#\theta^*} I_\iota^*$;

then I^* is a T-List for each $C[\![\theta_\iota]\!]$;

hence by 2.3, I^* is a T-List for $CEval[\![\theta^*]\!]$;

now suppose I is a Terminator for $CEval[\![\theta^*]\!], \eta', \chi^*, \sigma_0$;

then $\forall \eta, \theta$, $Develop(C[\![\theta]\!]\eta\theta)\chi^*\sigma_0$

$= Develop(CEval[\![\theta^*]\!]\eta_0\theta)\chi^*\sigma_0$

where $\eta_0 = \eta[\theta/\mathbf{cont}/I]$

$= \chi_1^* \hat{} Develop(\eta_0[\![\mathbf{cont}]\!][\![I]\!])(\chi^* \dagger \nu)\sigma_1$

where $\chi_1^* = Develop(CEval[\![\theta^*]\!]\eta'\theta_t)\chi^*\sigma_0$

$= \chi_1^* \hat{} Develop\theta(\chi^* \dagger \nu)\sigma_1$

$= Develop(C[\![\theta]\!]\eta'\theta_t)\chi^*\sigma_0 \hat{} Develop\theta(\chi^* \dagger \nu)\sigma_1$;

hence ✓ is a Terminator for $C[\![\theta]\!], \eta', \chi^*, \sigma_0$;

but any other Terminator for $CEval[\![\theta^*]\!]$ etc is easily shown to be a Terminator also for $C[\![\theta]\!]$ etc.

Hence $I^* \backslash \langle I \rangle$ is a T-List for $C[\![\theta]\!]$.

Case ⟦**while** E **do** Θ_1⟧ :

let I_1^*=*Jumps*⟦E⟧

and I_2^*=*Jumps*⟦Θ_1⟧

and I^*=*Jumps*⟦**while** E **do** Θ_1⟧=$I_1^* \hat{} I_2^*$;

then I^* is a T-List for *R*⟦E⟧ and *C*⟦Θ_1⟧ ;

If ? or some $I \triangleleft I^*$ is a Terminator for *R*⟦E⟧,η',χ^*,σ_0

then it is also for *C*⟦Θ⟧,η',χ^*,σ_0 ;

So suppose ✓ is a Terminator for *R*⟦E⟧,η',χ^*,σ_0 ;

then $\exists\varepsilon$, $\forall\eta,\theta$,

Develop(*C*⟦Θ⟧$\eta\theta$)$\chi^*\sigma_0$

= *Develop*(*R*⟦E⟧η{$\lambda\varepsilon.\varepsilon\rightarrow$*C*⟦$\Theta_1$⟧$\eta${*Break*$\theta'$},$\theta$})$\chi^*\sigma_0$

= $\chi'^* \hat{}$*Develop*($\varepsilon\rightarrow$*C*⟦Θ_1⟧η{*Break*θ'},θ)$\chi_1^*\sigma_1$

where χ'^*=*Develop*(*R*⟦E⟧$\eta'\kappa_t$)$\chi^*\sigma_0$;

If ε=*False*

then *Develop*(*C*⟦Θ⟧$\eta\theta$)$\chi^*\sigma_0$

= $\chi'^* \hat{}$*Develop*$\theta\chi_1^*\sigma_1$

= *Develop*(*C*⟦Θ⟧$\eta'\theta_t$)$\chi^*\sigma_0 \hat{}$*Develop*$\theta\chi_1^*\sigma_1$;

so ✓ is a Terminator

If ε=*True*

then *Develop*(*C*⟦Θ⟧$\eta\theta$)$\chi^*\sigma_0$

= $\chi'^* \hat{}$*Develop*(*C*⟦Θ_1⟧η(*Break*θ'))$\chi_1^*\sigma_1$;

so, as above, if ? or some $I \triangleleft I^*$ is a Terminator for

C⟦Θ_1⟧,η',χ_1^*,σ_1, then it is also for *C*⟦Θ⟧,η',χ^*,σ_0 ;

so suppose ✓ is a Terminator for *C*⟦Θ_1⟧,η',$\chi_1^*\sigma_1$;

ie $\forall\eta,\theta$, *Develop*(*C*⟦Θ⟧$\eta\theta$)$\chi^*\sigma_0$

= $\chi'^* \hat{} \chi''^* \hat{}$*Develop*(*Break*$\theta'$)$\chi_2^*\sigma_2$

where χ''^*=*Develop*(*C*⟦Θ_1⟧$\eta'\theta_t$)$\chi_1^*\sigma_1$

$= \chi'^{*}\hat{}\chi''^{*}\hat{}\langle \lambda\sigma.\sigma\rangle \hat{}\mathit{Develop}(C[\![\Theta]\!]\eta\theta)\chi_3^{*}\sigma_3$;

but $\#(\mathit{Develop}(C[\![\Theta]\!]\eta'\theta_t)\chi_3^{*}\sigma_3)$

$\leq \#(\mathit{Develop}(C[\![\Theta]\!]\eta'\theta_t)\chi^{*}\sigma_0) - \#\langle \lambda\sigma.\sigma\rangle$

$\leq \nu-1$;

hence by inductive hypothesis,

✓, ? or some $I \triangleleft I^{*}$ is a Terminator for

$C[\![\Theta]\!],\eta',\chi_3^{*},\sigma_3$;

and $C[\![\Theta]\!],\eta',\chi^{*},\sigma_0$ have the same Terminator;

Hence ✓, ? or some $I \triangleleft I^{*}$ is a Terminator for $C[\![\Theta]\!],\eta',\chi^{*},\sigma_0$;
ie I^{*} is a T-List for $C[\![\Theta]\!]$.

Case [[escape I]] :

For any η,θ,

$\mathit{Develop}(C[\![\Theta]\!]\eta\theta)\chi^{*}\sigma_0 = \mathit{Develop}(\eta[\![\mathbf{cont}]\!][\![I]\!])\chi^{*}\sigma_0$;

and $\mathit{Develop}(C[\![\Theta]\!]\eta'\theta_t)\chi^{*}\sigma_0 = \langle\rangle$;

so $\mathit{Develop}(C[\![\Theta]\!]\eta\theta)\chi^{*}\sigma_0$

$= \mathit{Develop}(C[\![\Theta]\!]\eta'\theta_t)\chi^{*}\sigma_0\hat{}\mathit{Deveop}(\eta[\![\mathbf{cont}]\!][\![I]\!])\chi^{*}\sigma_0$;

ie I is a Terminator for $C[\![\Theta]\!],\eta',\chi^{*},\sigma_0$;

so $\langle I\rangle = \mathit{Jumps}[\![\Theta]\!]$ is a T-List for $C[\![\Theta]\!]$.

Case [[B]] :

let $\varepsilon = B[\![B]\!]$;

then $\forall\eta,\kappa$,

$\mathit{Develop}(E[\![E]\!]\eta\kappa)\chi^{*}\sigma_0$

$= \mathit{Develop}(\kappa\varepsilon)\chi^{*}\sigma_0$

$= \mathit{Develop}(E[\![E]\!]\eta'\kappa_t)\chi^{*}\sigma_0\hat{}\mathit{Develop}(\kappa\varepsilon)\chi^{*}\sigma_0$;

so ✓ is a Terminator for $E[\![E]\!],\eta',\chi^{*},\sigma_0$;

so $\langle\rangle$ is a T-List for $E[\![E]\!]$.

Case ⟦I⟧ :

similar to case ⟦B⟧ with $\varepsilon=\eta'[\![var]\!][\![I]\!]=\eta[\![var]\!][\![I]\!]$.

All other cases are easily proved from the structural inductive hypothesis and theorem 2.3, using arguments similar to those of 3.2.1.

QED

3.3 Regularity of ELM

The next important property is that all the semantic equations define functions which are Regular. As with Consistency, this property will be shown to be satisfied without the constraints of any syntactic conditions. The proof of Regularity is in fact only required as part of the process of discovering which primitive operations are used, as described in the next section; indeed, the results of this section and the next are inextricably linked, as one case of theorem 3.3 depends on results in theorem 3.4, so that a completely rigorous proof would require the inductions of 3.3 and 3.4 to be done together. The two sections have, however, been treated separately for the sake of simplicity and clarity.

The importance of theorem 3.3 (which otherwise is very similar to, but weaker than, the results of the last section) lies in the predicates which constrain the values produced by expressions and declarations. These play an important role in the classification of operations in the next section. Some of them simply ensure that the semantics is sufficiently correct to

produce values in the expected domains - eg, any value produced by *L* is a L-value (or location) and any value produced by *R* is a R-value (specifically, an element of V). Others embody important properties of the language, viz, any location produced by a declaration is a newly claimed one - local to the process which evaluates the declaration; and any location produced as the result of evaluating an expression is either local to the process, or is to be found in a certain small, syntactically defined area of the environment. More precisely, the syntactic function $IdeVal_\epsilon$[Exp→Ide*] is defined (see Appendix 1) which gives a list of identifiers one of which will denote any non-local location which may be the result of the given expression.

The predicate *Denoted* introduced in 2.5 is useful in expressing some of the predicates. Its definition in ELM is, of course, simply:

*Denoted*δIη = (δ=η⟦var⟧⟦I⟧)

Lemma 3.3.1:

For any $\tau_0 \epsilon [E \to T]$ and $\tau, \tau' \epsilon [[S \to S] \to T]$,

Lv is Regular wrt τ, τ', $(\lambda\varepsilon.\varepsilon\epsilon L \Rightarrow \tau_0\varepsilon)$, $(\lambda\varepsilon.\varepsilon\epsilon L \wedge \tau_0\varepsilon)$.

Proof

Take any $\chi^*, \sigma_0, \varepsilon$ s.t. $\tau^*\chi^*$ and $(\varepsilon\epsilon L \Rightarrow \tau_0\varepsilon)$;

If $\varepsilon\epsilon L$

then $\tau_0\varepsilon$

and ∀κ, $Lv\kappa\varepsilon = \kappa\varepsilon$;

so $Develop(Lv\kappa\varepsilon)\chi^*\sigma_0 = \chi'^* \hat{} Develop(\kappa\varepsilon)\chi^*\sigma_0$

where $\chi'^* = Develop(Lv\kappa_t\varepsilon)\chi^*\sigma_0 = \langle\rangle$;

and ε satisfies $(\varepsilon\in L \wedge \tau_0\varepsilon)$;

and $\tau'^*\chi'^*$.

If $\varepsilon\notin L$

then $\forall\kappa$, $Lv\kappa\varepsilon = Error$ which is independent of κ ;

so $Develop(Lv\kappa\varepsilon)\chi^*\sigma_0$ is independent of κ ;

Thus the conditions of Regularity are satisfied.

QED

Lemma 3.3.2:

For any τ,τ',τ_0 s.t. for any $\alpha\in L$, $\beta\in B$, $\tau'(CheckFetch\alpha\beta)$,

Rv is Regular wrt τ, τ', τ_0, $(\lambda\varepsilon.\varepsilon\in V)$.

Proof

Take any $\chi^*,\sigma_0,\varepsilon$;

If $\varepsilon\in L$

then let $\varepsilon' = Fetch\varepsilon\sigma_0 \in V=B$;

then $\forall\kappa$, $Rv\kappa\varepsilon\sigma_0 = \langle CheckFetch\varepsilon\varepsilon', \kappa\varepsilon'\rangle$;

so $\forall\kappa$, $Develop(Rv\kappa\varepsilon)\chi^*\sigma_0$

$= \langle CheckFetch\varepsilon\varepsilon'\rangle \hat{} Develop(\kappa\varepsilon')(\chi^*\dagger 1)\sigma_1$

where $\sigma_1 = \chi_1(CheckFetch\varepsilon\varepsilon'\sigma_0)$

$= Develop(Rv\kappa_t\varepsilon)\chi^*\sigma_0 \hat{} Develop(\kappa\varepsilon')(\chi^*\dagger 1)\sigma_1$;

and $\tau'^*(Develop(Rv\kappa_t\varepsilon)\chi^*\sigma_0)$;

If $\varepsilon\in B$

then $\forall\kappa$, $Rv\kappa\varepsilon = \kappa\varepsilon$;

so $\forall\kappa$, $Develop(Rv\kappa\varepsilon)\chi^*\sigma_0 = \chi'^* \hat{} Develop(\kappa\varepsilon)\chi^*\sigma_0$

where $\chi'^* = Develop(Rv\kappa_t\varepsilon)\chi^*\sigma_0 = \langle\rangle$;

so $\tau'^*\chi'^*$;

Otherwise $Rv\kappa\varepsilon = Error$ which is independent of κ ;

hence $Develop(Rv\kappa\varepsilon)\chi^*\sigma_0$ is independent of κ ;

In any case, the conditions of Regularity are satisfied.

QED

Theorem 3.3:

For any Θ, E, Δ and η,

if $\pi = \eta[\![\mathbf{proc}]\!]$

and $\tau = Invariant(\pi)$

and $\tau' = Subvariant(\pi)$

and $\tau_{0\eta} = \lambda\alpha.((\exists I \in IdeVal[\![E]\!], Denoted\alpha I\eta) \vee (\eta[\![\mathbf{proc}]\!] \leq Owner\alpha))$,

then $C[\![\Theta]\!]\eta$ is Regular wrt τ,τ' ;

$E[\![E]\!]\eta$ is Regular wrt $\tau,\tau',\tau_{1\eta} = (\lambda\varepsilon.\varepsilon\in L \Rightarrow \tau_{0\eta}\varepsilon)$;

$L[\![E]\!]\eta$ is Regular wrt $\tau,\tau',\tau_{2\eta} = (\lambda\varepsilon.\varepsilon\in L \wedge \tau_{0\eta}\varepsilon)$;

$R[\![E]\!]\eta$ is Regular wrt $\tau,\tau',\tau_{3\eta} = (\lambda\varepsilon.\varepsilon\in V)$;

$D[\![\Delta]\!]\eta$ is Regular wrt $\tau,\tau',\tau_{4\eta}$

where $\tau_{4\eta} = (\lambda\varepsilon^*.\forall\iota,((\varepsilon_\iota\in L \wedge \eta[\![\mathbf{proc}]\!] \leq Owner\varepsilon_\iota) \vee \varepsilon_\iota \in V) \wedge$

$\forall\iota,\iota',(\varepsilon_\iota = \varepsilon_{\iota'}, \not\perp \Rightarrow \iota = \iota'))$.

Proof

Results for *L* and *R* follow immediately from that for *E* and 3.3.1 and 3.3.2.

For the others, take any χ^*,σ_0 s.t. $\tau^*\chi^*$, and proceed by induction on $\nu = \#(Develop(C[\![\Theta]\!]\eta\theta_t)\chi^*\sigma_0)$ etc and on the syntactic structure.

Note that $\tau_{0\eta},\tau_{1\eta},\tau_{2\eta},\tau_{3\eta},\tau_{4\eta}$ are H-Monotonic, so corollary 2.4 is applicable.

Case $[\![E_1 := E_2]\!]$:

from induction and corollary 2.4,

$LREval\langle [\![E_1]\!], [\![E_2]\!] \rangle \eta$ is Regular wrt

$\tau, \tau', (\lambda\langle \alpha,\beta \rangle . \tau_{2\eta}\alpha \wedge \tau_{3\eta}\beta)$;

and $\gamma = \lambda\theta.\lambda\langle \alpha,\beta \rangle . Update\alpha\beta\theta$ satisfies

$\forall \chi^*, \sigma_0, \langle \alpha,\beta \rangle$ s.t. $\tau^*\chi^*$

$\forall\theta$, $Develop(\gamma\theta\langle \alpha,\beta \rangle)\chi^*\sigma_0$

$= Develop(Update\alpha\beta\theta)\chi^*\sigma_0$

$= \langle Put\alpha\beta \rangle \hat{} Develop\theta(\chi^* \dagger 1)\sigma_1$

where $\sigma_1 = \chi_1(Put\alpha\beta\sigma_0)$

$= Develop(\gamma\theta_t\langle \alpha,\beta \rangle)\chi^*\sigma_0 \hat{} Develop\theta(\chi^* \dagger 1)\sigma_1$

and $\tau'(Put\alpha\beta)$;

so γ is Regular wrt τ,τ',τ'' for any τ'' ;

Hence by 2.4.5, $C[\![\theta]\!]\eta$ is Regular wrt τ,τ'.

Case $[\![\S I\ \theta^*\ \S I]\!]$:

$\forall\theta$, the inductive hypothesis and corollary 2.4 show that

$CEval[\![\theta^*]\!]\eta_\theta$ is Regular wrt τ,τ'

where $\eta_\theta = \eta[\theta/\mathbf{cont}/I]$;

ie $\forall\theta, \chi^*, \sigma_0$ s.t. $\tau^*\chi^*$,

either $\forall\theta'$, $Develop(CEval[\![\theta^*]\!]\eta_\theta\theta')\chi^*\sigma_0$

$= \chi'^* \hat{} Develop\theta'\chi_1^*\sigma_1$ (1)

where $\chi'^* = Develop(CEval[\![\theta^*]\!]\eta_\theta\theta_t)\chi^*\sigma_0$

and $\chi_1^* = \chi^* \dagger (\#\chi'^*)$

and $\sigma_1 = Consequence(CEval[\![\theta^*]\!]\eta_\theta\theta_t)\chi^*\sigma_0$

and $\tau'^*\chi'^*$;

or $Develop(CEval[\![\theta^*]\!]\eta_\theta\theta')\chi^*\sigma_0$ is independent of θ' ; (2)

but by 3.2 and 2.3, $CEval[\![\theta^*]\!]$ has a T-List I^* ;

so take any χ^*, σ_0 s.t. $\tau^*\chi^*$

and let $\eta' = \eta[\lambda I.\theta_t/\mathbf{cont}]$;

then there are four cases to consider:

i) $\checkmark$ is a Terminator for $CEval[\![\theta^*]\!],\eta',\chi^*,\sigma_0$:

in which case, since $\eta'=\eta_\theta[\lambda I.\theta_t/\mathbf{cont}]$ for any θ ,

$\forall\theta,\theta'$, $Develop(CEval[\![\theta^*]\!]\eta_\theta\theta')\chi^*\sigma_0 = \chi'^*\hat{}Develop\theta'\chi_1^*\sigma_1$

which is not independent of θ' ;

hence (1) holds, so $\tau'^*\chi'^*$;

in particular,

$\forall\theta$, $Develop(C[\![\theta]\!]\eta\theta)\chi^*\sigma_0 = Develop(CEval[\![\theta^*]\!]\eta_\theta\theta)\chi^*\sigma_0$

$= \chi'^*\hat{}Develop\theta\chi_1^*\sigma_1$

and $Develop(C[\![\theta]\!]\eta\theta_t)\chi^*\sigma_0=\chi'^*$

ii) $I\lhd I^*$ and I is a Terminator for $CEval[\![\theta^*]\!],\eta',\chi^*,\sigma_0$:

then $\forall\theta$, $Develop(C[\![\theta]\!]\eta\theta)\chi^*\sigma_0$

$= Develop(CEval[\![\theta^*]\!]\eta_\theta\theta)\chi^*\sigma_0$

$= \chi'^*\hat{}Develop\theta\chi_1^*\sigma_1$

where $\chi'^*=Develop(CEval[\![\theta^*]\!]\eta'\theta_t)\chi^*\sigma_0$;

and $Develop(C[\![\theta]\!]\eta\theta_t)\chi^*\sigma_0 = \chi'^*$.

Unfortunately, the Regularity of $CEval[\![\theta^*]\!]$ is insufficient, in this case, to show that $\tau'^*\chi'^*$.

However, the results of 3.4 applied to each $C[\![\theta_l]\!]$, together with 2.5, show that χ'^* satisfies $(Subvariant(\eta[\![\mathbf{proc}]\!]))$ as required.

iii) $I'\lhd I^*$ is a Terminator for $CEval[\![\theta^*]\!],\eta',\chi^*,\sigma_0$ $(I'\neq I)$:

then $Develop(C[\![\theta]\!]\eta\theta)\chi^*\sigma_0$

$= Develop(CEval[\![\theta^*]\!]\eta_\theta\theta)\chi^*\sigma_0$

$= \chi'^*\hat{}Develop(\eta_\theta[\![\mathbf{cont}]\!][\![I']\!])\chi_1^*\sigma_1$

$= \chi'^*\hat{}Develop(\eta[\![\mathbf{cont}]\!][\![I']\!])\chi_1^*\sigma_1$

which is independent of θ.

iv) ? is a Terminator for $CEval[\![\theta^*]\!],\eta',\chi^*,\sigma_0$:

then $\forall\theta$, $Develop(C[\![\theta]\!]\eta\theta)\chi^*\sigma_0 = Develop(CEval[\![\theta^*]\!]\eta'\theta_t)\chi^*\sigma_0$

which is independent of θ.

Case [**while** E **do** θ_1] :

Again, the recursive nature of the semantic equations for this case necessitates the use of induction on ν.

Since by structural induction,

$R[\![E]\!]\eta$ is Regular wrt τ,τ',τ_1 for some τ_1,

either $\forall\theta$, $Develop(C[\![\theta]\!]\eta\theta)\chi^*\sigma_0 = Develop(R[\![E]\!]\eta\kappa_1)\chi\ \sigma_0^*$

$= Develop(R[\![E]\!]\eta\kappa_t)\chi^*\sigma_0$

where $\kappa_1=(\lambda\varepsilon.\varepsilon\rightarrow C[\![\theta_1]\!]\eta(Break(C[\![\theta]\!]\eta\theta)),\theta)$;

so $Develop(C[\![\theta]\!]\eta\theta)\chi^*\sigma_0$ is independent of θ ;

or $\exists\varepsilon$ s.t. $\tau_1\varepsilon$ and

$\forall\theta$, $Develop(C[\![\theta]\!]\eta\theta)\chi^*\sigma_0$

$= Develop(R[\![E]\!]\eta\kappa_1)\chi^*\sigma_0$

$= \chi'^*\hat{}Develop(\kappa_1\varepsilon)\chi_1^*\sigma_1$

where $\chi'^*=Develop(R[\![E]\!]\eta\kappa_t)\chi^*\sigma_0$

etc.

so if $\varepsilon=False$

then $Develop(C[\![\theta]\!]\eta\theta)\chi^*\sigma_0$

$= \chi'^*\hat{}Develop\theta\chi_1^*\sigma_1$

$= Develop(C[\![\theta]\!]\eta\theta_t)\chi^*\sigma_0\hat{}Develop\theta\chi_1^*\sigma_1$;

while if $\varepsilon=True$

then $\forall\theta$, $Develop(C[\![\theta]\!]\eta\theta)\chi^*\sigma_0$

$= \chi'^*\hat{}Develop(C[\![\theta_1]\!]\eta\theta_1)\chi_1^*\sigma_1$

where $\theta_1=Break(C[\![\theta]\!]\eta\theta)$;

but $C[\![\theta_1]\!]\eta$ is Regular wrt τ,τ' ;

either $Develop(C[\![\Theta_1]\!]\eta\theta_1)\chi_1^*\sigma_1$ is independent of θ_1 ;

hence $Develop(C[\![\Theta]\!]\eta\theta)\chi^*\sigma_0$ is independent of θ ;

or $\forall\theta_1$, $Develop(C[\![\Theta_1]\!]\eta\theta_1)\chi_1^*\sigma_1$

$\chi''^* \hat{} Develop\,\theta_1\chi_2^*\sigma_2$

where $\chi''^* = Develop(C[\![\Theta_1]\!]\eta\theta_t)\chi_1^*\sigma_1$;

so $\forall\theta$, $Develop(C[\![\Theta]\!]\eta\theta)\chi^*\sigma_0$

$= \chi'^* \hat{} \chi''^* \hat{} \langle\lambda\sigma.\sigma\rangle \hat{} Develop(C[\![\Theta]\!]\eta\theta)\chi_3^*\sigma_3$

where $\chi_3^* = \chi_2^* \dagger 1$;

but then $\#Develop(C[\![\Theta]\!]\eta\theta)\chi_3^*\sigma_3 \leq \nu-1$;

so by inductive hypothesis,

either $Develop(C[\![\Theta]\!]\eta\theta)\chi_3^*\sigma_3$ is independent of θ ;

hence so is $Develop(C[\![\Theta]\!]\eta\theta)\chi^*\sigma_0$;

or $Develop(C[\![\Theta]\!]\eta\theta)\chi_3^*\sigma_3$

$= Develop(C[\![\Theta]\!]\eta\theta_t)\chi_3^*\sigma_3 \hat{} Develop\,\theta\chi_4^*\sigma_4$;

so $Develop(C[\![\Theta]\!]\eta\theta)\chi^*\sigma_0$

$= Develop(C[\![\Theta]\!]\eta\theta_t)\chi^*\sigma_0 \hat{} Develop\,\theta\chi_4^*\sigma_4$;

furthermore χ'^*, χ''^*, $\langle\lambda\sigma.\sigma\rangle$ and

$Develop(C[\![\Theta]\!]\eta\theta_t)\chi_3^*\sigma_3$ all satisfy τ'^* ;

so $\tau'^*(Develop(C[\![\Theta]\!]\eta\theta_t)\chi^*\sigma_0)$.

Case [[escape I]] :

$\forall\theta$, $C[\![\Theta]\!]\eta\theta = \eta[\![\mathbf{cont}]\!][\![I]\!]$

which is independent of θ ;

so $\forall\theta$, $Develop(C[\![\Theta]\!]\eta\theta)\chi^*\sigma_0$ is independent of θ .

Case [[let Δ in Θ_1]] :

follows immediately from structural induction and 2.4.5, the conditions imposed on the results of *D* being irrelevant.

Case ⟦B⟧ :

let $\varepsilon = B[\![B]\!]$;

since $\varepsilon \in B$, it certainly satisfies $\tau_{1\eta}$;

and $\forall\kappa$, $Develop(E[\![E]\!]\eta\kappa)\chi^*\sigma_0 = \chi'^* \hat{} Develop(\kappa\varepsilon)\chi^*\sigma_0$

where $\chi'^* = Develop(E[\![E]\!]\eta\kappa_t)\chi^*\sigma_0 = \langle\rangle$;

so trivially $\tau'^*\chi'^*$.

Case ⟦I⟧ :

let $\varepsilon = \eta[\![var]\!][\![I]\!]$;

then since $IdeVal[\![E]\!] = \langle I \rangle$,

ε satisfies $\tau_{0\eta}$ and therefore also $\tau_{1\eta}$;

and Regularity follows as in previous case.

Case $[\![E_1\ \Omega\ E_2]\!]$:

by structural induction and corollary 2.4

and definition of *IdeVal*,

$REval\langle [\![E_1]\!], [\![E_2]\!] \rangle \eta$ is Regular wrt $\tau, \tau', \tau^*_{3\eta}$;

but for any ε^*,

$\varepsilon = W[\![\Omega]\!]\varepsilon^* \in B$ and so satisfies $\tau_{1\eta}$;

hence $(\lambda\kappa.\lambda\varepsilon^*.\kappa(W[\![\Omega]\!]\varepsilon^*))$ is Regular wrt $\tau, \tau', \tau^*_{3\eta}, \tau_{1\eta}$;

so by 2.4.5, $E[\![E]\!]\eta$ is Regular wrt $\tau, \tau', \tau_{1\eta}$.

Case ⟦Θ **before** E_1⟧ :

Case $[\![E_1 \rightarrow E_2, E_3]\!]$:

Result follows imediately from structural induction and definition of *IdeVal*.

Case ⟦**let** Δ **in** E_1⟧ :

by structural induction, $D[\![\Delta]\!]\eta$ is Regular wrt $\tau, \tau', \tau_{4\eta}$;

so either $Develop(D[\![\Delta]\!]\eta\kappa')\chi^*\sigma_0$ is independent of κ' ;

hence $Develop(E[\![E]\!]\eta\kappa)\chi^*\sigma_0$ is independent of κ;

or $\exists\varepsilon^*$ s.t. $\tau_{4\eta}\varepsilon^*$ and

$\forall\kappa$, $Develop(E[\![E]\!]\eta\kappa)\chi^*\sigma_0 = \chi'^*\hat{}Develop(E[\![E_1]\!]\eta'\kappa)\chi_1^*\sigma_1$

where $\chi'^*=Develop(D[\![\Delta]\!]\eta\kappa_t')\chi^*\sigma_0$

and $\eta'=\eta[\varepsilon^*/var/I[\![\Delta]\!]]$;

now let τ_0',τ_1' be as $\tau_{0\eta},,\tau_{1\eta}$, but referring to E_1 ;

then by induction, $E[\![E_1]\!]\eta'$ is Regular wrt τ,τ',τ_1' ;

but if $I\lessdot IdeVal[\![E_1]\!]$,

then either $I\lessdot IdeVal[\![E]\!]$

or $I\lessdot I[\![\Delta]\!]$;

but if $I\lessdot I[\![\Delta]\!]$

then $\alpha=\eta'[\![var]\!][\![I]\!]\lessdot\varepsilon^*$;

hence $\tau_{4\eta}\langle\alpha\rangle$;

ie if $\alpha\epsilon L$, then $\eta[\![proc]\!]\leq Owner\alpha$;

otherwise, $\alpha=\eta'[\![var]\!][\![I]\!]=\eta[\![var]\!][\![I]\!]$;

and also $\eta'[\![proc]\!]=\eta[\![proc]\!]$;

hence $(\tau_0'\alpha\wedge\alpha\epsilon L)\Rightarrow$

$((\exists I\lessdot IdeVal[\![E]\!], Denoted\alpha I\eta)\vee(\eta[\![proc]\!]\leq Owner\alpha))$;

ie $\tau_1'\alpha \Rightarrow \tau_{1\eta}\alpha$;

hence $E[\![E_1]\!]\eta'$ is Regular wrt $\tau,\tau',\tau_{1\eta}$;

hence by 2.4.5, $E[\![E]\!]\eta$ is Regular wrt $\tau,\tau',\tau_{1\eta}$.

Case $[\![I = E]\!]$:

by structural induction, $R[\![E]\!]\eta$ is Regular wrt $\tau,\tau',\tau_{3\eta}$;

let $\gamma=\lambda\kappa'.\lambda\beta.NewLoc\pi(\lambda\alpha.Update\alpha\beta(\kappa'\langle\alpha\rangle))$

where $\pi=\eta[\![proc]\!]$;

Take any β s.t. $\tau_{3\eta}\beta$;

If $StoreFull\pi\sigma_0$

then $\forall\kappa'$, $\gamma\kappa'\beta\sigma_0=\langle CheckFull\pi,\theta_t\rangle$;

so $Develop(\gamma\kappa'\beta)\chi^*\sigma_0 = \langle CheckFull\pi\rangle$
which is independent of κ' ;

otherwise let $\alpha = New\pi\sigma_0$;
then $Owner\alpha = \pi$;
so $Develop(\gamma\kappa'\beta)\chi^*\sigma_0$
$= \langle \chi'\rangle \hat{} Develop(Update\alpha\beta(\kappa'\langle \alpha\rangle))(\chi^* \dagger 1)\sigma_1$
where $\chi' = CheckNew\pi\alpha$
and $\sigma_1 = (\chi_1 \circ \chi')\sigma_0$
$= \langle \chi'\rangle \hat{} \langle \chi''\rangle \hat{} Develop(\kappa'\langle \alpha\rangle)(\chi^* \dagger 2)\sigma_2$
where $\chi'' = Put\alpha\beta$
and $\sigma_2 = (\chi_2 \circ \chi'')\sigma_1$;
so $\tau'\chi'$ and $\tau'\chi''$;
so $\tau'^*(Develop(\gamma\kappa'_t\beta)\chi^*\sigma_0)$;

Hence result follows from 2.4.5.

Case $[\![I \equiv E]\!]$:

$R[\![E]\!]\eta$ is Regular wrt $\tau, \tau', \tau_{3\eta}$;
so either $\exists\varepsilon$ s.t. $\tau_{3\eta}\varepsilon$ ie $\varepsilon\epsilon$ V
and $\forall\kappa$, $Develop(R[\![E]\!]\eta\kappa)\chi^*\sigma_0 = \chi'^* \hat{} Develop(\kappa\varepsilon)\chi_1^*\sigma_1$
where $\chi'^* = Develop(R[\![E]\!]\eta\kappa_t)\chi^*\sigma_0$;
so $\forall\kappa'$, $Develop(D[\![\Delta]\!]\eta\kappa')\chi^*\sigma_0 = \chi'^* \hat{} Develop(\kappa'\langle \varepsilon\rangle)\chi_1^*\sigma_1$
where $\chi'^* = Develop(D[\![\Delta]\!]\eta\kappa'_t)\chi^*\sigma_0$;

or $Develop(R[\![E]\!]\eta\kappa)\chi^*\sigma_0$ is independent of κ ;
so $Develop(D[\![\Delta]\!]\eta\kappa')\chi^*\sigma_0$ is independent of κ' .

Case ⟦Δ_1 and Δ_2⟧ :

By induction and corollary 2.4,

$DEval\langle D⟦\Delta_1⟧, D⟦\Delta_2⟧\rangle\eta$ is Regular

wrt $\tau,\tau',(\lambda(\varepsilon_1^*,\varepsilon_2^*).\tau_{4\eta'},\varepsilon_1^* \wedge \tau_{4\eta''},\varepsilon_2^*)$

where $\eta'=\eta[\pi_1/\mathbf{proc}]$

and $\eta''=\eta[\pi_2/\mathbf{proc}]$

and $\eta⟦\mathbf{proc}⟧<\pi_1,\pi_2$ and $\pi_1 \gtrless \pi_2$;

so for $\alpha_1 \lessdot \varepsilon_1^*, \alpha_2 \lessdot \varepsilon_2^*$,

$\pi_1 \leq Owner\alpha_1$ and $\pi_2 \leq Owner\alpha_2$;

so $\alpha_1 \neq \alpha_2$;

and by H-Monotonicity of $\tau_{4\eta}$,

$\tau_{4\eta}(\varepsilon_1^* \hat{} \varepsilon_2^*)$;

ie $D⟦\Delta⟧$ is Regular wrt $\tau,\tau',\tau_{4\eta}$.

QED

3.4 Operations Produced by ELM

The last property required of the semantics of ELM concerns what basic operations are produced. Section 2.5 introduced the ten permitted types of operation and described the Sanction condition.

The model has been designed so that individual processes never interact except either by the process result mechanism (the ⟦res⟧ component of the state), which is carefully controlled so that operations on the same result position happen in a well defined order, or by trying to access the same storage location. To detect the latter syntactically, two sets of functions are needed, $IdeL \in [[\mathbf{Com+Exp+Dec}] \rightarrow \mathbf{Ide}^*]$ which gives a list of identifiers denoting all the locations used in L-mode in

a section of program, and *IdeR* and *RIdeR* (with the same functionality) which give corresponding lists for locations used in R-mode. Since one of the characteristics of ELM is that it allows no sharing, checking for Miscibility is simply a matter of verifying that the lists produced by these functions for different (potentially parallel) commands have no element in common, unless it occurs only in the R-mode list of both.

The following theorem shows that these two sets of functions achieve the desired effect.

Theorem 3.4:

For any η, χ^*, σ satisfying:

for each I, $\eta[\![\mathbf{cont}]\!][\![I]\!]$ is either θ_t or *Error*

and $(Invariant(\eta[\![\mathbf{proc}]\!]))^*\chi^*$;

then

any $\chi \leftarrow Develop(C[\![\Theta]\!]\eta\theta_t)\chi^*\sigma$ is Sanctioned

by $\eta, IdeL[\![\Theta]\!], IdeR[\![\Theta]\!]$;

any $\chi \leftarrow Develop(E[\![E]\!]\eta\kappa_t)\chi^*\sigma$ by $\eta, IdeL[\![E]\!], IdeR[\![E]\!]$;

any $\chi \leftarrow Develop(L[\![E]\!]\eta\kappa_t)\chi^*\sigma$ by $\eta, IdeL[\![E]\!], IdeR[\![E]\!]$;

any $\chi \leftarrow Develop(R[\![E]\!]\eta\kappa_t)\chi^*\sigma$ by $\eta, IdeL[\![E]\!], RIdeR[\![E]\!]$;

and any $\chi \leftarrow Develop(D[\![\Delta]\!]\eta\kappa'_t)\chi^*\sigma$ by $\eta, IdeL[\![\Delta]\!], IdeR[\![\Delta]\!]$.

Proof

by induction on $\nu = \#Develop(C[\![\Theta]\!]\eta\theta_t)\chi^*\sigma$ and on syntactic structure.

Most cases follow from the inductive hypothesis by the following kind of argument:

$C[\![\Theta]\!]\eta$ etc is Regular (by 3.3);

hence each operation χ produced by Θ is produced by one of its components;

it is therefore Sanctioned by η and the lists found by

applying *IdeL* etc to one of the components;

but *IdeL*⟦Θ⟧ and *IdeR*⟦Θ⟧ are obtained by concatenating such lists;

hence χ is Sanctioned by η,*IdeL*⟦Θ⟧,*IdeR*⟦Θ⟧ .

let $\tau_{0\eta}$ etc be as in theorem 3.3, and consider the cases which do not fit into the pattern described above.

Case ⟦E_1 := E_2⟧ :

by Regularity of *L*⟦E_1⟧η and *R*⟦E_2⟧ and by 2.4,

either $Develop(C⟦\Theta⟧\eta\theta_t)\chi^*\sigma$

$= Develop(LREval⟨⟦E_1⟧,⟦E_2⟧⟩\eta\kappa'_t)\chi^*\sigma$

$\hat{}\,Develop(Update\alpha\beta\theta_t)\chi_1^*\sigma_1$

where $\tau_{2\eta}\alpha$ and $\tau_{3\eta}\beta$

$= Develop(LREval⟨⟦E_1⟧,⟦E_2⟧⟩\eta\kappa'_t)\chi^*\sigma\hat{}\langle Put\alpha\beta\rangle$;

or $Develop(C⟦\Theta⟧\eta\theta_t)\chi^*\sigma$

$= Develop(LREval⟨⟦E_1⟧,⟦E_2⟧⟩\eta\kappa'_t)\chi^*\sigma$;

so in either case, if $\chi \triangleleft Develop(C⟦\Theta⟧\eta\theta_t)\chi^*\sigma$

either $\chi=(Put\alpha\beta)$

where $\alpha\in L$

and since $\tau_{2\eta}\alpha$,

either $\exists I \triangleleft IdeVal⟦E_1⟧ \leq IdeL⟦\Theta⟧$ s.t. $Denoted\alpha I\eta$

or $\eta⟦\mathbf{proc}⟧ \leq Owner\alpha$;

ie χ is Sanctioned by η,*IdeL*⟦Θ⟧,*IdeR*⟦Θ⟧ ;

or $\chi \triangleleft Develop(LREval⟨⟦E_1⟧,⟦E_2⟧⟩\eta\kappa'_t)\chi^*\sigma$;

but by inductive hypothesis,

$\forall\chi',\pi',\eta',\chi'^*,\sigma'$ s.t. $(Invariant\langle\pi'\rangle)^*\chi'^*\ \eta'=\eta[\pi'/\mathbf{proc}]$

$\chi' \triangleleft Develop(L⟦E_1⟧\eta'\kappa_t)\chi'^*\sigma'$

$\Rightarrow$ χ' is Sanctioned by η',*IdeL*⟦E_1⟧,*IdeR*⟦E_1⟧

$\Rightarrow$ χ' is Sanctioned by η',*IdeL*⟦Θ⟧,*IdeR*⟦Θ⟧ ;

and similarly,

$\chi' \lhd$ *Develop*(*R*⟦E_2⟧$\eta'\kappa_t$)$\chi^*\sigma$

$\Rightarrow$ χ' is Sanctioned by η',*IdeL*⟦E_2⟧,*RIdeR*⟦E_2⟧

$\Rightarrow$ χ' is Sanctioned by η',*IdeL*⟦Θ⟧,*IdeR*⟦Θ⟧ ;

so by 2.5, χ is Sanctioned by η,*IdeL*⟦Θ⟧,*IdeR*⟦Θ⟧ .

Case ⟦§I Θ^* §I⟧ :

let $\eta'=\eta[\theta_t/\mathbf{cont}/I]$;

by inductive hypothesis,

if $\chi \lhd$ *Develop*(*C*⟦Θ_1⟧$\eta_1'\theta_t$)$\chi_1^*\sigma_1$

where $\eta_1'=\eta'[\pi_1/\mathbf{proc}]$

and (*Invariant*(π_1))$^*\chi_1^*$

then χ is Sanctioned by η_1',*IdeL*⟦Θ_1⟧,*IdeR*⟦Θ_1⟧ ;

so χ is Sanctioned by η_1',*IdeL*⟦Θ⟧,*IdeR*⟦Θ⟧ ;

hence by 2.5,

if $\chi \lhd$ *Develop*(*C*⟦Θ⟧$\eta\theta_t$)$\chi^*\sigma$ = *Develop*(*CEval*⟦Θ^*⟧$\eta'\theta_t$)$\chi^*\sigma$

then χ is Sanctioned by η',*IdeL*⟦Θ⟧,*IdeR*⟦Θ⟧ ;

but Sanction does not depend on the ⟦**cont**⟧ component;

so χ is Sanctioned by η,*IdeL*⟦Θ⟧,*IdeR*⟦Θ⟧ .

Case ⟦**while** E **do** Θ_1⟧ :

Again this case requires the double induction.

R⟦E⟧η and *C*⟦Θ_1⟧η are Regular, so by the usual argument,

if $\chi \lhd$ *Develop*(*C*⟦Θ⟧$\eta\theta_t$)$\chi^*\sigma$

then one of the following holds:

i) $\chi \lhd$ *Develop*(*R*⟦E⟧$\eta\kappa_t$)$\chi^*\sigma$;

so χ is Sanctioned by η,*IdeL*⟦E⟧,*RIdeR*⟦E⟧ ;

and therefore by η,*IdeL*⟦Θ⟧,*IdeR*⟦Θ⟧ ;

ii) $\chi \lhd \mathit{Develop}(C[\![\Theta_1]\!]\eta\theta_t)\chi_1^*\sigma_1$;

so χ is Sanctioned by $\eta, \mathit{IdeL}[\![\Theta_1]\!], \mathit{IdeR}[\![\Theta_1]\!]$;

and therefore by $\eta, \mathit{IdeL}[\![\Theta]\!], \mathit{IdeR}[\![\Theta]\!]$;

iii) $\chi = (\lambda\sigma.\sigma)$ which is Sanctioned by anything;

iv) $\chi \lhd \mathit{Develop}(C[\![\Theta]\!]\eta\theta_t)\chi_2^*\sigma_2$

where $\#\mathit{Develop}(C[\![\Theta]\!]\eta\theta_t)\chi_2^*\sigma_2 \leq \nu - 1$;

hence by ν induction,

χ is Sanctioned by $\eta, \mathit{IdeL}[\![\Theta]\!], \mathit{IdeR}[\![\Theta]\!]$.

Case [[escape I]] :

$C[\![\Theta]\!]\eta\theta_t$ is either θ_t or *Error* ;

so only possible χ is *SetError*

which is Sanctioned by anything.

Case [[let Δ in Θ_1]] :

by 3.3 and inductive hypothesis,

if $\chi \lhd \mathit{Develop}(C[\![\Theta]\!]\eta\theta_t)\chi^*\sigma$

and $\langle \pi_1, \pi_2 \rangle = \mathit{NewProcs}(\eta[\![\mathbf{proc}]\!])2\sigma$;

then a) $\chi = (\mathit{CheckProcs}(\eta[\![\mathbf{proc}]\!])\langle \pi_1, \pi_2 \rangle)$;

which is Sanctioned by η, etc.

or b) $\chi \lhd \mathit{Develop}(D[\![\Delta]\!]\eta\kappa_t')\chi^*\sigma$;

in which case χ is Sanctioned

by $\eta[\pi_1/\mathbf{proc}], \mathit{IdeL}[\![\Delta]\!], \mathit{IdeR}[\![\Delta]\!]$;

and therefore by $\eta, \mathit{IdeL}[\![\Theta]\!], \mathit{IdeR}[\![\Theta]\!]$;

or c) $\exists \varepsilon^*$ s.t. $\tau_{4\eta}\varepsilon^*$

and $\chi \lhd \mathit{Develop}(C[\![\Theta_1]\!]\eta'\theta_t)\chi_1^*\sigma_1$

where $\eta' = \eta[\varepsilon^*/\mathbf{var}/I[\![\Delta]\!]][\pi_2/\mathbf{proc}]$;

hence χ is Sanctioned by $\eta', \mathit{IdeL}[\![\Theta_1]\!], \mathit{IdeR}[\![\Theta_1]\!]$;

ie χ is a Permitted Operation and

1) if χ is $Put\alpha\beta$

then either $\exists I \prec IdeL[\![\Theta_1]\!]$ s.t. $Denoted\alpha I\eta'$

or $\eta'[\![\mathbf{proc}]\!] \leq Owner\alpha$;

ie one of the following holds:

a) $I \prec IdeL[\![\Theta_1]\!]$ and $I \prec \not\equiv[\![\Delta]\!]$;

so $\alpha = \eta'[\![\mathbf{var}]\!][\![I]\!] \prec \varepsilon^*$;

so $\eta[\![\mathbf{proc}]\!] \leq \eta'[\![\mathbf{proc}]\!] \leq Owner\alpha$;

b) $I \prec IdeL[\![\Theta_1]\!] \setminus I[\![\Delta]\!]$;

so $I \prec IdeL[\![\Theta]\!]$;

and $\alpha = \eta'[\![\mathbf{var}]\!][\![I]\!] = \eta[\![\mathbf{var}]\!][\![I]\!]$;

c) $\eta[\![\mathbf{proc}]\!] \leq \eta'[\![\mathbf{proc}]\!] \leq Owner\alpha$;

hence either $\exists I \prec IdeL[\![\Theta]\!]$ s.t. $Denoted\alpha I\eta$

or $\eta[\![\mathbf{proc}]\!] \leq Owner\alpha$;

2) if χ is $CheckFetch\alpha\beta$

then either $\exists I \prec IdeR[\![\Theta_1]\!]$ s.t. $Denoted\alpha I\eta'$

or $\eta'[\![\mathbf{proc}]\!] \leq Owner\alpha$;

and as above, this implies

either $\exists I \prec IdeR[\![\Theta]\!]$ s.t. $Denoted\alpha I\eta$

or $\eta[\![\mathbf{proc}]\!] \leq Owner\alpha$;

3) if χ is any of the other permitted forms,

then since $\eta[\![\mathbf{proc}]\!] \leq \eta'[\![\mathbf{proc}]\!]$,

the conditions of Sanction carry over,

ie χ is Sanctioned by $\eta, IdeL[\![\Theta]\!], IdeR[\![\Theta]\!]$;

The results for *L* and *R* follow from that for *E* since:

Either $Develop(L[\![E]\!]\eta\kappa_t)\chi^*\sigma = Develop(E[\![E]\!]\eta\kappa_t)\chi^*\sigma$

or $Develop(L[\![E]\!]\eta\kappa_t)\chi^*\sigma = Develop(E[\![E]\!]\eta\kappa_t)\chi^*\sigma\hat{}\langle SetError\rangle$;

so if $\chi \lhd \mathit{Develop}(L[\![E]\!]\eta\kappa_t)\chi^*\sigma$

then either $\chi \lhd \mathit{Develop}(E[\![E]\!]\eta\kappa_t)\chi^*\sigma$

or $\chi = (\mathit{SetError})$;

and in either case χ is Sanctioned

by $\eta, \mathit{IdeL}[\![E]\!], \mathit{IdeR}[\![E]\!]$;

Also,

either $\mathit{Develop}(R[\![E]\!]\eta\kappa_t)\chi^*\sigma = \mathit{Develop}(E[\![E]\!]\eta\kappa_t)\chi^*\sigma$

or $\mathit{Develop}(R[\![E]\!]\eta\kappa_t)\chi^*\sigma = \mathit{Develop}(E[\![E]\!]\eta\kappa_t)\chi^*\sigma \hat{} \langle \mathit{CheckFetch}\alpha\beta \rangle$

where $\alpha \in L$ and $\tau_{0\eta}\alpha$;

so if $\chi \lhd \mathit{Develop}(R[\![E]\!]\eta\kappa_t)\chi^*\sigma$

then either $\chi \lhd \mathit{Develop}(E[\![E]\!]\eta\kappa_t)\chi^*\sigma$;

in which case χ is Sanctioned by $\eta, \mathit{IdeL}[\![E]\!], \mathit{IdeR}[\![E]\!]$;

or $\chi = (\mathit{CheckFetch}\alpha\beta)$;

which is Sanctioned by $\eta, \mathit{IdeL}[\![E]\!], \mathit{IdeVal}[\![E]\!]$

since $\tau_{0\eta}\alpha$;

so in either case,

χ is Sanctioned by $\eta, \mathit{IdeL}[\![E]\!], \mathit{RIdeR}[\![E]\!]$

since $\mathit{RIdeR}[\![E]\!] = \mathit{IdeR}[\![E]\!] \hat{} \mathit{IdeVal}[\![E]\!]$.

Case $[\![B]\!]$:

Case $[\![I]\!]$:

$\mathit{Develop}(E[\![E]\!]\eta\kappa_t)\chi^*\sigma = \langle\rangle$;

so no such χ exists.

Case $[\![$ **let** Δ **in** $E_1]\!]$

is similar to case $[\![$ **let** Δ **in** $\Theta_1]\!]$.

Case ⟦ I = E⟧ :

by 3.3, *R*⟦E⟧ is Regular ;

so if $\chi \lessdot \mathit{Develop}(D[\![\Delta]\!]\eta\kappa_t')\chi^*\sigma$

then either $\chi \lessdot \mathit{Develop}(R[\![E]\!]\eta\kappa_t)\chi^*\sigma$;

in which case χ is Sanctioned

by η,*IdeL*⟦E⟧,*RIdeR*⟦E⟧ ;

and therefore by η,*IdeL*⟦Δ⟧,*IdeR*⟦Δ⟧ ;

or $\chi \lessdot \mathit{Develop}(\mathit{NewLoc}\pi(\lambda\alpha.\mathit{Update}\alpha\beta\theta_t))\chi_1^*\sigma_1$

for some $\beta \in B$

where $\pi = \eta[\![\mathrm{proc}]\!]$;

so χ is one of (*CheckFull*π), (*CheckNew*πα), (*Put*αβ)

(where $\alpha = \mathit{New}\pi\sigma_1$ ie *Owner*α=π) ;

all of which are Sanctioned by η and any ide lists.

QED

3.5 The Syntactic Conditions for ELM

All that now remains to be done is to piece together the results of previous sections, and to formulate the required syntactic conditions.

After what has been said in 3.2 and 3.4, the conditions needed are fairly obvious: to ensure JumpFreeness, the result of applying *Jumps* to each of the program sections must be an empty list; to ensure Miscibility, the identifier lists introduced in 3.4 must have no element in common, unless it is only in the R-mode lists. The second of these is formulated in the form of the predicates *CSeparable* etc below.

To complete the picture, two constraints have to be imposed on the environments. The first, Share-Free, guarantees the

assertion made earlier that ELM does not allow two identifiers to share the same location. The second, P-Strict, embodies another significant property of the semantic definition of ELM used, namely that all locations available to a section of program through the environment were claimed by a process incomparable with the current one.

Definitions

$$Disjoint I_1^* I_2^* = (\forall I_1 \triangleleft I_1^*,\ \forall I_2 \triangleleft I_2^*,\ I_1 \neq I_2)$$

$$CSeparable[\![\Theta^*]\!] =$$

$$\forall \iota, \iota' \leq \#\Theta^*,\ (\iota = \iota' \vee Disjoint(IdeL[\![\Theta_\iota]\!])(IdeL[\![\Theta_{\iota,}]\!] \hat{}\ IdeR[\![\Theta_{\iota,}]\!]))$$

$$RSeparable[\![E^*]\!] =$$

$$\forall \iota, \iota' \leq \#E^*,\ (\iota = \iota' \vee Disjoint(IdeL[\![E_\iota]\!])(IdeL[\![E_{\iota,}]\!] \hat{}\ RIdeR[\![E_{\iota,}]\!])$$

and similarly for *LRSeparable*

Definition

η is *Share-Free* if

$$\forall I_1, I_2 \in Ide$$

$$\eta[\![\text{var}]\!][\![I_1]\!] = \eta[\![\text{var}]\!][\![I_2]\!] \in L \Rightarrow I_1 = I_2$$

η is *P-Strict* if

$$\forall I \in Ide$$

$$\eta[\![\text{var}]\!][\![I]\!] = \alpha \in L \Rightarrow Owner\alpha \gtrsim \eta[\![\text{proc}]\!]$$

Lemma 3.5.1:

If *CSeparable*⟦θ*⟧ and $\theta^{\pi*}=\langle \pi^*, \lambda\pi_1.C⟦\theta_1⟧\eta[\pi_1/\mathbf{proc}]\theta_t\rangle$

and $\forall\pi_1<\pi^*$, $\eta⟦\mathbf{proc}⟧<\pi_1$

and η is Share-Free and P-Strict

then $\chi^{*\pi*}=\langle \pi^*, \lambda\pi.Detail\theta^{\pi*}\zeta\pi\sigma\rangle$ is Miscible

or some component of some $\chi^{*\pi*}\pi$ is *SetError* or *CheckFull*π' .

Proof

by 2.4.1, $\exists\chi_1', \chi_1^*$ s.t. $(Invariant(\pi_1))^*\chi_1^*$ and

$\chi^{*\pi*}=\langle \pi^*, \lambda\pi_1.Develop(C⟦\theta_1⟧\eta[\pi_1/\mathbf{proc}]\theta_t)\chi_1^*(\chi_1'\sigma)\rangle$;

take any $\pi_1, \pi_1' <\pi^*$ s.t. $\pi_1\neq\pi_1'$;

and let χ,χ' be any component of

$\chi^{*\pi*}\pi_1, \chi^{*\pi*}\pi_1'$ respectively;

Case 1: χ is $(Put\alpha\beta_1)$ and χ' is $(Put\alpha\beta_2)$ or $(CheckFetch\alpha\beta_2)$;

then a) $\alpha=\eta⟦\mathbf{var}⟧⟦I⟧$ for some $I<IdeL⟦\theta_1⟧$;

and $\alpha=\eta⟦\mathbf{var}⟧⟦I'⟧$ for $I'<IdeL⟦\theta_1,⟧\hat{}IdeR⟦\theta_1,⟧$;

but by Separability, $I\neq I'$;

which contradicts Share-Freeness of η ;

or b) $\alpha=\eta⟦\mathbf{var}⟧⟦I⟧$ for some $I<IdeL⟦\theta_1⟧$;

and $\pi_1, \leq Owner\alpha$;

but $\eta⟦\mathbf{proc}⟧<\pi_1'$;

so $\eta⟦\mathbf{proc}⟧<Owner\alpha$;

which contradict the P-Strict condition;

or c) $\pi_1\leq Owner\alpha$;

and $\alpha=\eta⟦\mathbf{var}⟧⟦I⟧$ for some $I<IdeL⟦\theta_1,⟧\hat{}IdeR⟦\theta_1,⟧$;

which leads to the same contradiction as b;

or d) $\pi_1\leq Owner\alpha$

and $\pi_1, \leq Owner\alpha$;

which is impossible by tree structure of P

and incomparability of π_1 and π_1' ;

Hence this case is impossible.

Case 2: χ is $(Signal\pi\varepsilon_1)$

and χ' is $(Signal\pi\varepsilon_2)$ or $(CheckResults\pi^*\varepsilon^*)$ where $\pi < \pi^*$;

then Sanction implies that

$\pi_1 < \pi$ and $\pi_1', < \pi$;

which is impssible as in Case 1d.

Case 3: χ is $(CheckNew\pi\alpha_1)$ and χ' is $(CheckNew\pi\alpha_2)$;

Again, Sanction implies the impossible conditions

$\pi_1 \leq \pi$ and $\pi_1', \leq \pi$.

Case 4: χ is $(CheckProcs\pi\pi_1^*)$ and χ' is $(CheckProcs\pi\pi_2^*)$;

which is impossible as in case 3.

Case 5: one of χ, χ' is $(SetError)$ or $(CheckFull\pi')$.

Case 6: any other combination;

in which case, by 2.5.1, χ and χ' commute.

Thus $\chi^{*\pi*}$ is Miscible unless case 5 holds.

QED

Lemma 3.5.2:

$(\lambda\Theta^*\eta.\mathit{CSeparable}[\![\Theta^*]\!] \wedge \forall\iota,\#\mathit{Jumps}[\![\Theta_\iota]\!]=0)$

is a P-Test for C wrt ($\lambda\eta.\eta$ is Share-Free and P-Strict).

Proof

For any Θ^*,η,

If $\mathit{CSeparable}[\![\Theta^*]\!]$ then by 3.5.1,

$\chi^{*\pi^*}=\langle \pi^*,\lambda\pi.\mathit{Detail}\langle \pi^*,\lambda\pi_\iota.C[\![\Theta_\iota]\!]\eta[\pi_\iota/\mathbf{proc}]\theta_t\rangle \zeta\pi\sigma\rangle$

either is Miscible

or satisfies $\mathit{ErrorSet}(\mathit{Apply}(\mathit{Mix}\chi^{*\pi^*}\zeta)\sigma)$;

by 3.1, each $C[\![\Theta_\iota]\!]\eta[\pi_\iota/\mathbf{proc}]$ preserves Consistency;

and by 3.3, for each ι,

if $\mathit{Jumps}[\![\Theta_\iota]\!]=\langle\rangle$

then $\langle\rangle$ is a T-List for $C[\![\Theta_\iota]\!]$.

QED

These lemmas lead (at last!) to the main theorem, which asserts that parallel execution of an ELM program, subject to the syntactic constraints described above, does indeed lead to "correct" results - ie the same as produced by serial execution. The ubiquitous and troublesome possibility of error has to be allowed for, but for the purposes of this investigation it has been assumed that if an error is detected, then so long as an error is detected however the program is executed, nothing else matters! - a considerable simplification, but one without which the work would be unmanageable.

Theorem 3.5:

If η_s and η_p are Share-Free and P-Strict and satisfy

$\forall I \in \mathbf{Ide}$, $Expand(\eta_s[\![\mathbf{cont}]\!][\![I]\!])\sigma \approx Expand(\eta_p[\![\mathbf{cont}]\!][\![I]\!])\sigma$

and $\eta_p = \eta_s[\eta_p[\![\mathbf{cont}]\!]/\mathbf{cont}]$,

and $Expand\theta_s\sigma \approx Expand\theta_p\sigma$

and $\forall\varepsilon$, $Expand(\kappa_s\varepsilon)\sigma \approx Expand(\kappa_p\varepsilon)\sigma$

and $\forall\varepsilon^*$, $Expand(\kappa_s'\varepsilon^*)\sigma \approx Expand(\kappa_p'\varepsilon^*)\sigma$

then $C_s[\![\theta]\!]\eta_s\theta_s$ and $C_p[\![\theta]\!]\eta_p\theta_p$ are X-Equivalent wrt σ;

and $L_s[\![E]\!]\eta_s\kappa_s$ and $L_p[\![E]\!]\eta_p\kappa_p$ are X-Equivalent wrt σ;

etc

where C_s, L_s and C_p, L_p etc denote the semantic functions using the serial and parallel forms of *CEval* etc and using the P-Test of 3.5.2.

Hence $\forall\theta,\sigma$ and Share-Free, P-Strict η,

$Outcome(C_s[\![\theta]\!]\eta\theta)\sigma = Outcome(C_p[\![\theta]\!]\eta\theta)\sigma$

or both sides have *ErrorSet* ;

and similarly for E, L, R, D.

Proof

by structural induction.

Case $[\![\,\S I\ \theta^*\ \$I\,]\!]$:

let $\eta_s' = \eta_s[\theta_s/\mathbf{cont}/I]$

and $\eta_p' = \eta_p[\theta_p/\mathbf{cont}/I]$;

so η_s' and η_p' satisfy the same hypotheses as η_s and η_p ;

so $Expand(C_s[\![\theta]\!]\eta_s\theta_s)\sigma$

$= Expand(Eval_3(\mathop{\text{⫯}}_{\iota} C_s[\![\theta_\iota]\!])\eta_s'\theta_s)\sigma$

$\approx Expand(Eval_3(\mathop{\text{⫯}}_{\iota} C_p[\![\theta_\iota]\!])\eta_p'\theta_p)\sigma$

by induction and 2.4.3

$\approx$ *Expand*$((\tau[\![\Theta^*]\!]\eta'_p \rightarrow Eval_4, Eval_3)(\not\!\!\downarrow_i C_p[\![\Theta_i]\!])\eta'_p\theta_p)\sigma$

$\approx$ *Expand*$(C_p[\![\Theta]\!]\eta_p\theta_p)\sigma$.

Case ⟦ **let** Δ **in** Θ_1 ⟧ :

let $\langle \pi_1, \pi_2 \rangle$ = *NewProcs*$(\eta[\![\textbf{proc}]\!])2\sigma$

and $\eta'_s = \eta_s[\pi_1/\textbf{proc}]$

and $\eta'_p = \eta_p[\pi_1/\textbf{proc}]$;

$D_s[\![\Delta]\!]\eta'_s$ and $D_p[\![\Delta]\!]\eta'_p$ are Regular wrt $\tau, \tau', \tau_{4\eta'}$ by 3.3 ;

(Note $\tau_{4\eta'}$ does not depend on ⟦ **cont** ⟧ component so is the same for η'_s and η'_p;)

and by inductive hypothesis,

$\forall \kappa', \sigma$, *Expand*$(D_s[\![\Delta]\!]\eta'_s\kappa')\sigma \approx$ *Expand*$(D_p[\![\Delta]\!]\eta'_p\kappa')\sigma$;

so as in 2.4.2 and 2.4.3, $D_s[\![\Delta]\!]\eta'_s$ and $D_p[\![\Delta]\!]\eta'_p$ satisfy the same alternative of Regularity;

moreover, by choosing κ' so that $(\lambda\varepsilon^*.$*Expand*$(\kappa'\varepsilon^*)\sigma)$ is an injective mapping from $\ddot{E}^*$ to $[S \rightarrow S]^*$,

eg $(\lambda\varepsilon^*\sigma.\langle$ *CheckResults*$\pi^*\varepsilon^*, \theta_t\rangle)$ for some π^* ,

it can be seen that, if they satisfy the first alternative, then it is with the same ε^* ;

hence either $\exists\varepsilon^*$ s.t. $\tau_{4\eta}\varepsilon^*$ and

Expand$(C_s[\![\Theta]\!]\eta_s\theta_s)\sigma$

$= \langle \chi \rangle$ ^*Expand*$(D_s[\![\Delta]\!]\eta'_s\kappa'_t)\sigma$^*Expand*$(C_s[\![\Theta_1]\!]\eta''_s\theta_s)\sigma_2$

where $\sigma_2 =$ *Outcome*$(D_s[\![\Delta]\!]\eta'_s\kappa'_t)(\chi\sigma)$

and $\pi = \eta_s[\![\textbf{proc}]\!] = \eta_p[\![\textbf{proc}]\!]$

and $\chi =$ *CheckProcs*$\pi\langle \pi_1, \pi_2 \rangle$

and $\eta''_s = \eta_s[\varepsilon^*/\textbf{var}/I[\![\Delta]\!]][\pi_2/\textbf{proc}]$;

but for all $I \in$ **Ide**,

if $I \lessdot I[\![\Delta]\!]$ then $\alpha = \eta''_s[\![\textbf{var}]\!][\![I]\!] \lessdot \varepsilon^*$;

so $\pi_1 \leq$ *Owner*α ;

ie since $\pi_1 \gtrless \pi_2$,

$\pi_2 \lesseqgtr Owner\alpha$;

otherwise $\alpha = \eta_s''[\![var]\!][\![I]\!] = \eta_s[\![var]\!][\![I]\!]$;

so since η_s is P-Strict,

$\pi \lesseqgtr Owner\alpha$;

and $\pi < \pi_2$;

then $\pi_2 \lesseqgtr Owner\alpha$;

so η_s'' is P-Strict;

also, for any $I_1, I_2 \in Ide$, s.t. $I_1 \neq I_2$,

and for $\iota = 1,2$, $\alpha_\iota = \eta_s''[\![var]\!][\![I_\iota]\!] \in L$,

if $I_1, I_2 \lessdot I[\![\Delta]\!]$

then α_1 and α_2 are distinct elements of ε^* ;

hence $\alpha_1 \neq \alpha_2$;

if $I_1 \lessdot I[\![\Delta]\!]$ but $\sim I_2 \lessdot I[\![\Delta]\!]$ (or vice versa)

then $\alpha_1 \lessdot \varepsilon^*$ so $\pi < \pi_1 \leq Owner\alpha_1$;

but $\alpha_2 = \eta_s[\![var]\!][\![I_2]\!]$ so $\sim \pi \leq Owner\alpha_2$;

so $\alpha_1 \neq \alpha_2$;

if $\sim I_1 \lessdot I[\![\Delta]\!]$ and $\sim I_2 \lessdot I[\![\Delta]\!]$

then $\alpha_\iota = \eta_s[\![var]\!][\![I_\iota]\!]$;

so by Share-Freeness of η_s, $\alpha_1 \neq \alpha_2$;

hence η_s'' is Share-Free ;

and similarly for η_p'' ;

hence $Expand(C_s[\![\Theta_1]\!]\eta_s''\theta_s)\sigma_2 \approx Expand(C_p[\![\Theta_1]\!]\eta_p''\theta_p)\sigma_2$;

so $Expand(C_s[\![\Theta]\!]\eta_s\theta_s)\sigma \approx Expand(C_p[\![\Theta]\!]\eta_p\theta_p)\sigma$;

or $Expand(C_s[\![\Theta]\!]\eta_s\theta_s)\sigma$

$= Expand(D_s[\![\Delta]\!]\eta_s'\kappa_t')\sigma$

$\approx Expand(D_p[\![\Delta]\!]\eta_p'\kappa_t')\sigma$

$$\approx \mathit{Expand}(C_p[\![\Theta]\!]\eta_p\theta_p)\sigma \ .$$

QED

Compiler-usable conditions have now been found (lemma 3.5.2) for the very simple case of ELM. Although this language does not display great potential for parallelism, it demonstrates the sort of conditions necessary. It is worth noting the similarity and differences between the conditions derived here and those derived by others, although they are expressed in a somewhat different form. The concept of Separability corresponds to the basic conditions of independence as in eg [24], and could be extended to the slightly more sophisticated conditions of eg [8]; while the addition of the Jumps condition enables the present work to more generally applicable, rather than requiring certain common combinations of language constructions (eg blocks of assignment statements) to be picked out, and the analysis applied only to these. The conditions will now be extended, in the next two chapters, to cover some more useful constructions.

CHAPTER 4: A LANGUAGE WITH PROCEDURES

"'What fun!' said the Gryphon... 'It's all her fancy, that: they never executes nobody, you know.'"

Alice's Adventures in Wonderland: Chap 9.

The second language can now be developed, by adding facilities to ELM. It will be known as POPLAR (Procedure Oriented Programming Language, Allowing Recursion) since its main feature is that it allows functions to be defined and used. Subroutines could also be added, but they only complicate the equations, and add nothing, since a function can do all that a subroutine can, and has the added complexity of returning a result. The opportunity has also been taken to introduce a problem which is often (though not necessarily) associated with procedures, namely "sharing" - by which two different identifiers simultaneously denote the same location. The relative simplicity of the conditions found in the last chapter depends on the complete prohibition of this common phenomenon from ELM.

This chapter will be very similar in form to the last, and proofs concerning POPLAR will not be repeated where they are the same as, or similar to, their ELM counterparts.

4.0 Description of POPLAR

The syntax of POPLAR is very like that of ELM, but with a few additions. There is a new syntactic domain **Fun** of function declarations, and a subsidiary domain **Spe** for specification of parameter calling modes; also, the domains **Com** and **Exp** are extended to allow function declarations and calls. A function declaration consists of (i) an identifier to denote the function, (ii) a list of identifiers for the formal parameters, (iii) a list, equal in length to the last, of specifiers giving the modes of the parameters, and (iv) an expression forming the body of the function. Functions may be recursive, but for the sake of simplifying the already complicated semantics, mutually recursive functions are not included - they should not add any significant problems. Unlike simple values, functions may not be stored, but only denoted directly by identifiers - as is the case in most existing languages. The effect of allowing stored functions is that no syntactic conditions can possibly be found whenever such a function is involved, since a compiler has, in general, no means of telling what such a function may be doing. Other features such as the ability to pass functions out of the scopes of their free variables have been considered, but they make the conditions, already complicated, quite prohibitively so, thus, although interesting and significant features, they are excluded. Indeed, even the passing of functions as parameters of functions has been omitted for this reason.

Parameters may be evaluated either in L-mode or in R-mode, indicated respectively by the specifiers **lv** and **rv**; in neither case is a new location used, but the formal parameter identifier is made to denote the L- or R-value produced. There is no

point in allowing "call by value" where the actual parameter is evaluated in R-mode and assigned to a new location, since this is equivalent to a combination of other constructs which are allowed. Algol's "call by name" has been ignored as being far too outlandish and complicated, and most other variations in calling modes are too similar to these four to merit separate mention.

The semantic domains show rather more differences. There is inevitably a domain F of function values, which are not, as in many semantic systems, evaluated at the time of declaration and given a form like $[E^*\rightarrow K\rightarrow C]$, but are represented in an unprocessed, syntactic form comprising (i) the formal parameter list, (ii) the specifier list, (iii) the expression forming the body, (iv) a considerably reduced environment which consists only of the ⟦var⟧ component of the environment at the declaration of the function, and (v) a "generation" or declaration level whose form and purpose is described further below. This is done for several reasons. Firstly, the statement and proof of many of the theorems is thereby simplified, since the proof of a certain condition usually requires that any fuctions in the current environment also satisfy a similar condition. If the usual "evaluated" representation of functions is used this means that all the theorems have to be encumbered with complicated restrictions on environments (which are obviously going to be satisfied); whereas, if an unevaluated form is used, a simple induction allows the result to be proved with no unnecessary conditions.

The other main reason for this representation arises from a more fundamental problem. The mathematical theory underlying

this work has been only briefly mentioned, but one important consideration is that in order to define the semantic equations, many of which are highly recursive in nature, every function used must be continuous. This usually causes few difficulties, since most of the commonly used building blocks produce continuous results; but the important exception is the = operator which is not necessarily continuous. Even this has not caused any trouble yet, since the only use of equality in recursively defined equations has been in the Consistency-ensuring state transformations *CheckFetch* etc., in which it was applied to elements of the domains T, V, L, P and E all of which were flat lattices - on which equality is always continuous. POPLAR, however, expects E to include functions, and domains like $[E^* \rightarrow K \rightarrow C]$ are far from flat. The problem, very roughly, arises from the fact that domains like $[E^* \rightarrow K \rightarrow C]$ include all possible functions from E^* and K to C, including "infinitely" complex ones, whereas the only such functions which ever occur are those produced by algorithms which are in some sense finite - represented by a finite number of syntactic symbols. Such considerations suggest the sort of representation of functions described above, with the partial environment further purged to a free variable list - with a finite number of entries. It is beyond the scope of the present work to consider this problem in more detail, so having indicated the problem and the direction of its solution, it will hereafter be ignored - the only cases where function values would cause trouble having been disallowed for other reasons (see above).

The domains D and E are extended to include F as a summand,

but since stored functions are not permitted, V remains simply as B.

The other changes are concerned with the environment. Firstly, by the time a function is used, the denotations of some of its free variables may have been hidden by subsequent declarations of the identifiers involved, so that the function is liable to access locations not available from the current environment in the normal way. It is, however, useful to have a complete list of possible denotations of each identifier, so the ⟦var⟧ component of H becomes $[Ide \to D^*]$. Thus the normal denotation of an identifier I will now be given by $\eta⟦var⟧⟦I⟧\downarrow 1$, the remaining components never being used in the semantic equations, but being required in some of the results to be proved below. As will become apparent in section 4.4, this formulation has its snags, but it will suffice.

Secondly, a new domain G of "syntactic environments" is introduced to cope with the new features of POPLAR. It has three components - one to deal with sharing, and two relating to functions. (The same three components are also added to H, but no conversion will be specified; instead, when η, η_0, η' etc. are given, $\gamma, \gamma_0, \gamma'$ etc. will be taken, by convention, to denote their respective projections into G.) The point of these new components is that they contain purely syntactic information. Thus a compiler can produce and use them to check the conditions of a P-Test, since these conditions are no longer context-free, ie independent of the environment, as was the case with ELM. It is, though, important that the P-Test depends only on the syntactic environment.

The new components are as follows:

(i) The ⟦gen⟧ component is an integer which gives the current "generation" or declaration level. Since a function can only have free variables declared at the same level as itself or at an outer level, the concept of generation is important when checking the "correctness" of the syntactic environment, by avoiding the confusion caused by declarations which cause holes in the scopes of some of the free variables of a function.
(ii) The ⟦fn⟧ component gives, for each identifier and each generation, information about any function which may be denoted by that identifier at that generation. This information itself has four components - two free variable lists, one each for variables used in L- and R-mode (note that "Free Variable List", as used here, is a list of identifiers which may denote locations, and it includes free variables of any functions called within the body under consideration); the formal parameter list; and the parameter specification list.
(iii) The ⟦share⟧ component is a list of identifiers which may share a denoted location with another identifier.

A few new operators will be needed and are conveniently defined here:

For any $\xi_1, \xi_2 \in X = [Ide \rightarrow D^*]$

$\xi_1 \subseteq \xi_2$ if

$\forall I,\ \exists \nu,\ \xi_1[\![I]\!] = \xi_2[\![I]\!] \dagger \nu$

For any $I_1^*, I_2^* \in Ide^*$

$I_1^* \leq I_2^*$ if

$\forall I,\ I \prec I_1^* \Rightarrow I \prec I_2^*$

and $I_1^* \equiv I_2^*$ if $I_1^* \leq I_2^* \leq I_1^*$

ie if $\forall I,\ I \prec I_1^* \Leftrightarrow I \prec I_2^*$

The detailed semantics of POPLAR can be found in Appendix 2. The functions *C*, *E* and *I* are extended to include the new constructions, and new functions are introduced: $F\epsilon$[Fun→H→K'→C] which processes function declarations and uses $G\epsilon$[Fun→H→F] to dissect a single declaration and pack it up into a F-value; $S\epsilon$[Spe→Exp→H→K→C] which selects the appropriate function (*L* or *R*) for evaluating parameters; and $A\epsilon$[Exp→H→K→C] which evaluates an expression expecting to find a function value as result. A fifth form of *Eval* is used - *SEval* which has an extra argument Σ^* and applies $S[\![\Sigma_\iota]\!]$ to the ιth component of its next argument.

4.1 Consistency of POPLAR

There is very little to be done to show the Consistency of the commands produced in POPLAR. All the results of 3.1 hold. so it remains only to prove the result for *F* (which is trivial, since *F* immediately applies its continuation without doing any computation), for *A* (which is almost identical to the proof for *L*), and for the new syntactic categories.

Theorem 4.1:

If $\theta_0,\kappa_0,\kappa_0',\eta$ are ν-Consistent

then $C[\![\Theta]\!]\eta\theta_0$, $E[\![E]\!]\eta\kappa_0$, $L[\![E]\!]\eta\kappa_0$, $R[\![E]\!]\eta\kappa_0$, $A[\![E]\!]\eta\kappa_0$, $D[\![\Delta]\!]\eta\kappa_0'$ and $F[\![\Phi]\!]\eta\kappa_0'$ are ν-Consistent.

Proof

by induction on ν and on the syntactic structure.

Most cases are exactly as in Theorem 3.1.

The Consistency of $A[\![E]\!]\eta\kappa_0$ follows from that of $E[\![E]\!]\eta\kappa_0$ in exactly the same way as with *L*.

$F[\![\Phi]\!]\eta\kappa_0'=\kappa_0'\phi^*$ for some ϕ^*

so Consistency follows immediately.

Case [[**let** Φ **in** Θ_1]] :

Case [[**let** Φ **in** E_1]] :

The result follows immediately from the induction hypothesis and the observations that $LR[\![E_1]\!]$ preserves Consistency (just as *L*, *R*, and *A*) and that the Consistency of environments depends only on their [[**cont**]] components.

Case $[\![E_1[E^*]]\!]$:

Since this case involves using a syntactic element which is not a component of E_1, the structural induction is not sufficient, so as in the case of the **while** loop, the induction on ν is required.

The occurrence of *Break* in *ApplyFn* allows the inductive step to be made, in a similar manner to the **while** loop (see 3.1). The remainder of the proof follows easily from the structural induction.

QED

4.2 Termination in POPLAR

There is again very li tle to do to show that *Jumps* provides a T-List in POPLAR, since both the new constructs are JumpFree - declaration of a function because it has no computation to do, and application because jumps out of functions are prohibited in POPLAR. Were that not so, another component would have to be added to the ⟦fn⟧ part of the environment to indicate where any particular function might jump to; the information thus provided would then have to be incorporated by *Jumps* whenever a function was applied, just as *IdeL* and *IdeR* do (see section 4.4).

Theorem 4.2:

For any Θ, Δ, E,

Jumps⟦Θ⟧ is a T-List for C⟦Θ⟧ ;

Jumps⟦E⟧ is a T-List for E⟦E⟧, L⟦E⟧, R⟦E⟧, A⟦E⟧ ;

Jumps⟦Δ⟧ is a T-List for D⟦Δ⟧ .

Proof

Mostly as 3.2.

Result for A follows from that for E in the same way as for L and R.

Case ⟦let Φ in Θ_1⟧ :

$\forall \eta, \theta, \chi^*, \sigma,$

$Develop(C⟦\Theta⟧\eta\theta)\chi^*\sigma = Develop(C⟦\Theta_1⟧\eta_1\theta)\chi^*\sigma$

for some η_1 ;

but by inductive hypothesis,

$I^* = Jumps⟦\Theta_1⟧$ is a T-List for C⟦Θ_1⟧ ;

ie ✓, ?, or some $I \lhd I^*$ is a Terminator for

$C[\![\theta_1]\!], \eta_1[\lambda I.\theta_t/cont], \chi^*, \sigma$;

hence ✓, ?, or some $I \lhd I^*$ is a Terminator for

$C[\![\theta]\!], \eta[\lambda I.\theta_t/cont], \chi^*, \sigma$;

so I^* is a T-List for $C[\![\theta]\!]$.

Case $[\![$ **let** Φ **in** $E_1]\!]$

is similar.

Case $[\![E_1[E^*]]\!]$:

This case again requires double induction,

first on $\nu = \#(Develop(E[\![E]\!]\eta'\theta_t)\chi_0^*\sigma_0)$;

By structural induction,

$Jumps[\![E_1]\!]$ is a T-List for $A[\![E_1]\!]$;

hence either ? or some $I \lhd Jumps[\![E_1]\!] \leq Jumps[\![E]\!]$ is a Terminator

for $A[\![E_1]\!], \eta', \chi_0^*, \sigma_0$;

and hence for $E[\![E]\!], \eta', \chi_0^*, \sigma_0$;

or ✓ is a Terminator for $A[\![E_1]\!], \eta', \chi_0^*, \sigma_0$;

so $\exists\phi, \forall\eta, \kappa$, $Develop(A[\![E_1]\!]\eta\kappa)\chi_0^*\sigma_0 =$

$Develop(A[\![E_1]\!]\eta'\kappa_t)\chi_0^*\sigma_0 \hat{} Develop(\kappa\phi)\chi_1^*\sigma_1$;

so if $\#E^* \neq \#\phi[\![$ params$]\!]$

then $Develop(E[\![E]\!]\eta\kappa)\chi_0^*\sigma_0 =$

$Develop(A[\![E_1]\!]\eta'\kappa_t)\chi_0^*\sigma_0)\hat{}\langle SetError\rangle$;

so ? is a Terminator;

otherwise $Develop(E[\![E]\!]\eta\kappa)\chi_0^*\sigma_0 =$

$Develop(A[\![E_1]\!]\eta'\kappa_t)\chi_0^*\sigma_0\hat{}\langle\chi\rangle\hat{}\chi'^*$

where $\chi = (CheckProcs\pi\langle\pi_1, \pi_2\rangle)$

and $\pi = \eta'[\![$ proc$]\!]$

and $\eta_\iota = \eta[\pi_\iota/proc]$

and $\chi'^{*}=Develop(SEval[\![\Sigma^{*}]\!][\![E^{*}]\!]\eta_1\kappa')\chi_1^{*}\sigma_1$

where $\kappa'=(\lambda\delta^{*}.ApplyFn[\![E_1]\!]\eta_2\phi\delta^{*}\kappa)$;

but by structural induction,

$I_1^{*}=Jumps[\![E_1]\!]$ is a T-List for $L[\![E_1]\!]$ and $R[\![E_1]\!]$;

and hence for $S[\![\Sigma_1]\!][\![E_1]\!]$;

so by 2.3,

$\ast_{1} Jumps[\![E_1]\!]$ is a T-List for $SEval[\![\Sigma^{*}]\!][\![E^{*}]\!]$;

so either ? or some $I \triangleleft \ast_{1} Jumps[\![E_1]\!] \leq Jumps[\![E]\!]$ is

a Terminator for $E[\![E]\!],\eta',\chi_0^{*},\sigma_0$

or $Develop(E[\![E]\!]\eta\kappa)\chi_0^{*}\sigma_0 = \chi''^{*}\hat{}Develop(\kappa'\delta^{*})\chi_2^{*}\sigma_2$

for some δ^{*}

$= \chi''^{*}\hat{}\langle\lambda\sigma.\sigma\rangle\hat{}Develop(R[\![E']\!]\eta_3\kappa)\chi_3^{*}\sigma_3$

where $E'=\phi[\![\mathbf{body}]\!]$;

but $\#(Develop(R[\![E']\!]\eta_3\kappa)\chi_3^{*}\sigma_3)\leq\nu-1$;

so by ν induction,

$Jumps[\![E']\!]$ is a T-List for $R[\![E']\!]$;

but if some $I \triangleleft Jumps[\![E']\!]$ is a Terminator for

$R[\![E']\!],\eta_3',\chi_3^{*},\sigma_3$;

then $Develop(R[\![E']\!]\eta_3\kappa)\chi_3^{*}\sigma_3$

$= \chi_1''^{*}\hat{}Develop(\eta_3[\![\mathbf{cont}]\!][\![I]\!])\chi_4^{*}\sigma_4$

$= \chi_1''^{*}\hat{}Develop(Error)\chi_4^{*}\sigma_4$;

so ? is also a Terminator;

hence ✓ or ? is a Termiantor for $E[\![E]\!],\eta',\chi_0^{*},\sigma_0$;

Thus $Jumps[\![E]\!]$ is a T-List for $E[\![E]\!]$.

QED

4.3 Regularity of POPLAR

As in the last two sections, showing that the semantics of POPLAR produces Regular results requires very little addition to the corresponding proof for ELM. The main addition is the function *AIdeVal* which has a very similar purpose to *IdeVal*, except that it produces a list of identifiers which may denote function values rather than locations. Its definition is identical to that of *IdeVal* except for one case, reflecting the fact that a function application can return a location as result, but not another function. A new Regularity condition is added for $A[\![E]\!]$ which makes use of this new function, and the result for $E[\![E]\!]$ is correspondingly extended.

The changed structure of H results in a modified definition of *Denoted*:

$Denoted\delta I\eta = (\exists \iota \leq \#\eta[\![var]\!][\![I]\!] \text{ st } \delta = \eta[\![var]\!][\![I]\!]\downarrow\iota)$

Theorem 4.3:

For any Θ, E, Δ, η,

if $\tau, \tau', \tau_{0\eta}, \tau_{4\eta}$ are as in theorem 3.3

and $\tau'_{0\eta} = \lambda\phi.(\exists I \lhd AIdeVal[\![E]\!], Denoted\phi I\eta)$

then $C[\![\Theta]\!]\eta$ is Regular wrt τ, τ' ;

$E[\![E]\!]\eta$ is Regular wrt $\tau, \tau', \tau_{1\eta}$

where $\tau_{\iota\eta} = \lambda\varepsilon.((\varepsilon \in L \wedge \tau_{0\eta}\varepsilon) \vee (\varepsilon \in V) \vee (\varepsilon \in F \wedge \tau'_{0\eta}\varepsilon))$;

$L[\![E]\!]\eta$ is Regular wrt $\tau, \tau', \tau_{2\eta} = \lambda\varepsilon.(\varepsilon \in L \wedge \tau_{0\eta}\varepsilon)$;

$R[\![E]\!]\eta$ is Regular wrt $\tau, \tau', \tau_{3\eta} = \lambda\varepsilon.(\varepsilon \in V)$;

$A[\![E]\!]\eta$ is Regular wrt $\tau, \tau', \tau_{5\eta} = \lambda\varepsilon.(\varepsilon \in F \wedge \tau'_{0\eta}\varepsilon)$;

$D[\![\Delta]\!]\eta$ is Regular wrt $\tau, \tau', \tau_{4\eta}$.

Proof

Very similar to that of 3.3, but with the addition of

the new syntactic categories,

the possibility of values in F,

and the result for *A* ;

and with the slight change in the structure of H.

The result for *A* follows from that for *E*,

together with the fact that $(\lambda\kappa\varepsilon.\varepsilon\in F\to\kappa\varepsilon, Error)$ is Regular

wrt $\tau,\tau',\tau'',(\lambda\varepsilon.\varepsilon\in F\wedge\tau''\varepsilon)$ for any τ'' .

Case ⟦ I ⟧ :

let $\varepsilon=\eta$⟦ var ⟧⟦ I ⟧$\downarrow 1$;

then since I⊲*IdeVal*⟦ E ⟧ and I⊲*AIdeVal*⟦ E ⟧,

$\tau_{1\eta}\varepsilon$ is certainly true;

so, since $\forall\kappa$, *E*⟦ E ⟧$\eta\kappa=\kappa\varepsilon$, Regularity follows trivially.

Case ⟦ let Φ in E_1 ⟧ :

let $\phi^*=Fix(\lambda\phi^*.G$⟦ Φ ⟧$(\eta[\phi^*\hat{}/var/I$⟦ Φ ⟧$])(\eta$⟦ gen ⟧$+1))$

and $\eta'=NoteAbs$⟦ Φ ⟧$(\eta[\phi^*\hat{}/$var$/I$⟦ Φ ⟧$])(\eta$⟦ gen ⟧ $+1)$;

then *E*⟦ E ⟧$\eta\kappa=LR$⟦ E_1 ⟧$\eta'\kappa$;

hence either $Develop(E$⟦ E ⟧$\eta\kappa)\chi^*\sigma_0$ is independent of κ ;

or ∃ε st $(\varepsilon\in L\wedge\tau''_{0\eta},\varepsilon)\nu(\varepsilon\in V)$

and $\forall\kappa$, $Develop(E$⟦ E ⟧$\eta\kappa)\chi^*\sigma_0 = Develop(LR$⟦ E_1 ⟧$\eta'\kappa)\chi^*\sigma_0$

$\hat{}\ Develop(\kappa\varepsilon)\chi'^*\sigma_1$

where $\tau''_{0\eta}$ is $\tau_{0\eta}$ with E replaced by E_1 ;

but *IdeVal*⟦ E ⟧ = *IdeVal*⟦ E_1 ⟧

and η⟦ proc ⟧ = η'⟦ proc ⟧

and $\varepsilon\in L\nu\varepsilon\in V \Rightarrow$ (*Denoted*εI$\eta \Leftrightarrow$ *Denoted*εIη') ;

hence ε satisfies $(\varepsilon\in L\wedge\tau_{0\eta}\varepsilon)\nu(\varepsilon\in V)$;

ie $\tau_{1\eta}\varepsilon$.

Case $[\![E_1[E^*]]\!]$:

As in the case of the **while** loop, this case requires the second induction on $\nu=\#(Develop(E[\![E]\!]\eta\kappa_t)\chi^*\sigma_0)$ to cope with the occurrence of E' which is not a component of E. The presence of *Break* in *ApplyFn* is used in the same way to invoke the ν induction, otherwise it follows from the structural induction that:

$E[\![E]\!]\eta$ is Regular wrt $\tau,\tau',\tau_{3\eta}$

and therefore wrt $\tau,\tau',\tau_{1\eta}$.

QED

4.4 Operations produced by POPLAR

The results of the last three sections are fairly simple extensions of their ELM counterparts, but it is in examining the operations produced that the major changes arise. The first difficulty is that the functions *IdeL* and *IdeR* can no longer be context free, since when they come across a procedure application they must refer in some way to the definition of the procedure to discover its free variables. On the other hand, they must still be compiler evaluable. This is where the syntactic environments come in - they preserve sufficient information about any function denoted by a particular identifier for a compiler to check the P-Test conditions even when procedure applications are involved. For this reason, functions are not allowed to be stored, since then the

conditions cannot be checked until run time when it is discovered exactly which function is being applied.

It is vital, of course, that the syntactic part of the environment keeps in step with the denotation part. In fact, though, the syntactic part may contain information about more functions than are to be found in the denotation part. The reason for this is that when a procedure is applied, the denotation environment has to be pruned down to the state it was in at the declaration of the procedure; however, the compiler may not know precisely which procedure is being used (eg because of conditional expressions), so it cannot prune the syntactic part correspondingly, but has to make do with the same one. This means also that the syntactic environment must contain the required information also for the "declaration environment" of any procedure which may be applied. This is conveniently achieved by having all such "declaration environments" contained in the main environment, in the sense that all of the denotations of an identifier in the procedure's environment are its denotation also in the main one.

All this is summed up in the following definition. Note the use of the generation mentioned in the introduction to this chapter to prevent "expecting too much" of the syntactic environment, in eg the case where a declaration causes a hole in the scope of a free variable of a function declared at an outer level.

Definition

η is *Fn-Compatible* if

$\forall I,\phi=\langle I^*,\Sigma^*,E,\xi,\nu\rangle$ st *Denoted*$\phi I\eta$,

$IdeL[\![E]\!]\gamma\nu = \gamma[\![fn]\!][\![I]\!]\nu[\![lfree]\!]$,

$RIdeR[\![E]\!]\gamma\nu = \gamma[\![fn]\!][\![I]\!]\nu[\![rfree]\!]$,

$\Sigma^* = \gamma[\![fn]\!][\![I]\!]\nu[\![spe]\!]$,

$I^* = \gamma[\![fn]\!][\![I]\!]\nu[\![par]\!]$,

$\nu \le \eta[\![gen]\!]$,

$\xi \subseteq \eta[\![var]\!]$,

and $\forall I',\phi'=\langle I'^*,\Sigma'^*,E',\xi',\nu'\rangle$ st $\exists\iota'$, $\xi[\![I']\!]\downarrow\iota'=\phi'$,

$\xi' \subseteq \xi$ and $\nu' \le \nu$,

and $\forall I,\nu'>\eta[\![gen]\!]$, $\gamma[\![fn]\!][\![I]\!]\nu'=\langle\langle\rangle,\langle\rangle,\langle\rangle,\langle\rangle\rangle$.

This condition is complicated, so some lemmas are needed to show it is preserved by the kinds of transformations performed on environments in the semantic equations, before the main result of the section which shows that *IdeL* and *IdeR* still perform the function for which they were designed - to Sanction the primitive operations.

Lemma 4.4.1:

If η is Fn-Compatible,

then $\forall\nu'\ge\nu,\forall E$,

$FreeLVal[\![E]\!]\gamma\nu' \ge FreeLVal[\![E]\!]\gamma\nu$;

$FreeRVal[\![E]\!]\gamma\nu' \ge FreeRVal[\![E]\!]\gamma\nu$;

$IdeL[\![E]\!]\gamma\nu' \ge IdeL[\![E]\!]\gamma\nu$;

$IdeR[\![E]\!]\gamma\nu' \ge IdeR[\![E]\!]\gamma\nu$;

$RIdeR[\![E]\!]\gamma\nu' \ge RIdeR[\![E]\!]\gamma\nu$;

with equality if $\nu\ge\eta[\![gen]\!]$;

and similarly for Θ and Δ (*IdeL* and *IdeR* only).

Proof

from definition of FreeLVal,

$$FreeLVal[\![E]\!]\gamma\nu' \equiv$$

$$FreeLVal[\![E]\!]\gamma\nu \wedge \bigwedge_{\iota=\nu+1}^{\nu'} \left(\bigwedge_{I \triangleleft I^*} \gamma[\![fn]\!][\![I]\!]\iota[\![lfree]\!] \right)$$

where $I^* = AIdeVal[\![E]\!]$

$$\geq FreeLVal[\![E]\!]\gamma\nu ;$$

while if $i > \eta[\![gen]\!]$,

then $\left(\bigwedge_{I \triangleleft I^*} \gamma[\![fn]\!][\![I]\!]\iota[\![lfree]\!] \right) = ()$;

so equality holds if $\nu \geq \eta[\![gen]\!]$;

similarly for *FreeRVal* ;

Result for *IdeL* (*IdeR*) follows by structural induction using result for *FreeLVal* (*FreeRVal*) in case $[\![E_1[E^*]]\!]$;

Result for *RIdeR* is immediate from that for *IdeR* .

QED

Lemma 4.4.2:

If η is Fn-Compatible

and $\Phi = [\![I'[I'^*; \Sigma'^*] \equiv E']\!]$

and $\nu = \eta[\![gen]\!] + 1$

and $\phi^* = G[\![\Phi]\!](\eta[\phi^* \hat{} / var/I[\![\Phi]\!]])\nu$

and $\eta' = NoteAbs[\![\Phi]\!](\eta[\phi^* \hat{} / var/I[\![\Phi]\!]])\nu$

then for any $\iota < \nu$ and any E,

$FreeLVal[\![E]\!]\gamma'\iota = FreeLVal[\![E]\!]\gamma\iota$;

$FreeRVal[\![E]\!]\gamma'\iota = FreeRVal[\![E]\!]\gamma\iota$;

$IdeL[\![E]\!]\gamma'\iota = IdeL[\![E]\!]\gamma\iota$;

$IdeR[\![E]\!]\gamma'\iota = IdeR[\![E]\!]\gamma\iota$;

$RIdeR[\![E]\!]\gamma'\iota = RIdeR[\![E]\!]\gamma\iota$;

and for any E,

$$\mathit{FreeLVal}[\![E]\!]\gamma\nu \le \mathit{FreeLVal}[\![E]\!]\gamma'\nu \le \mathit{FreeLVal}[\![E]\!]\gamma\nu \hat{}\ \mathit{IdeL}[\![E']\!]\gamma\nu \ ;$$

$$\mathit{FreeRVal}[\![E]\!]\gamma\nu \le \mathit{FreeRVal}[\![E]\!]\gamma'\nu \le \mathit{FreeRVal}[\![E]\!]\gamma\nu \hat{}\ \mathit{RIdeR}[\![E']\!]\gamma\nu \ ;$$

$$\mathit{IdeL}[\![E]\!]\gamma\nu \le \mathit{IdeL}[\![E]\!]\gamma'\nu \le \mathit{IdeL}[\![E]\!]\gamma\nu \hat{}\ \mathit{IdeL}[\![E']\!]\gamma\nu \ ;$$

$$\mathit{IdeR}[\![E]\!]\gamma\nu \le \mathit{IdeR}[\![E]\!]\gamma'\nu \le \mathit{IdeR}[\![E]\!]\gamma\nu \hat{}\ \mathit{IdeR}[\![E']\!]\gamma\nu \ ;$$

$$\mathit{RIdeR}[\![E]\!]\gamma\nu \le \mathit{RIdeR}[\![E]\!]\gamma'\nu \le \mathit{RIdeR}[\![E]\!]\gamma\nu \hat{}\ \mathit{RIdeR}[\![E']\!]\gamma\nu \ ;$$

and hence η' is Fn-Compatible.

Proof

First part by structural induction on E.

for *FreeLVal* :

Case $[\![I]\!]$:

for $\iota<\nu$,

$$\begin{aligned}\mathit{FreeLVal}[\![E]\!]\gamma'\iota &= \gamma'[\![\mathbf{fn}]\!][\![I]\!]\iota[\![\mathbf{lfree}]\!] \\ &= \gamma[\![\mathbf{fn}]\!][\![I]\!]\iota[\![\mathbf{lfree}]\!] \\ &= \mathit{FreeLVal}[\![E]\!]\gamma\iota \ ;\end{aligned}$$

$$\begin{aligned}\text{and } \mathit{FreeLVal}[\![E]\!]\gamma'\nu &= \gamma'[\![\mathbf{fn}]\!][\![I]\!]\nu[\![\mathbf{lfree}]\!] \\ &\quad \gamma[\![\mathbf{fn}]\!][\![I]\!]\nu[\![\mathbf{lfree}]\!] \hat{}\ (I \lessdot I[\![\Phi]\!] \rightarrow \mathit{IdeL}[\![E']\!]\gamma\nu, \langle\rangle) \\ &= \mathit{FreeLVal}[\![E]\!]\gamma\nu \hat{}\ (I = I' \rightarrow \mathit{IdeL}[\![E']\!]\gamma\nu, \langle\rangle) \ .\end{aligned}$$

The other cases require simple applications of the induction.

Similarly for *FreeRVal*.

The results for *IdeL* etc. are easily proved by structural induction, using the results *FreeLVal* etc. in case $[\![E_1[E^*]]\!]$.

Second part:

Consider any I, ι st $\eta'[\![\mathbf{var}]\!][\![I]\!]\downarrow\iota = \phi = \langle I^*, \Sigma^*, E, \xi, \nu' \rangle \in F$;

then by definition of η',

either $\phi = \eta[\![\mathbf{var}]\!][\![I]\!]\downarrow\iota_1$ for some ι_1 ;

so $\nu' \leq \eta[\![\mathbf{gen}]\!] < \nu$;

so by first part,

$$\mathit{IdeL}[\![E]\!]\gamma'\nu' = \mathit{IdeL}[\![E]\!]\gamma\nu' \equiv \gamma[\![\mathbf{fn}]\!][\![I]\!]\nu'[\![\mathbf{lfree}]\!] \equiv \gamma'[\![\mathbf{fn}]\!][\![I]\!]\nu'[\![\mathbf{lfree}]\!] ;$$

and similarly for *RIdeR* ;

also $\Sigma^* = \gamma[\![\mathbf{fn}]\!][\![I]\!]\nu'[\![\mathbf{spe}]\!] - \gamma'[\![\mathbf{fn}]\!][\![I]\!]\nu'[\![\mathbf{spe}]\!]$;

and $I^* = \gamma[\![\mathbf{fn}]\!][\![I]\!]\nu'[\![\mathbf{par}]\!] - \gamma'[\![\mathbf{fn}]\!][\![I]\!]\nu'[\![\mathbf{spe}]\!]$;

and $\nu' < \nu = \eta'[\![\mathbf{gen}]\!]$;

and $\xi \subseteq \eta[\![\mathbf{var}]\!] \subseteq \eta'[\![\mathbf{var}]\!]$;

and $\forall I'', \iota''$, st $\xi[\![I'']\!]\downarrow\iota'' = \langle I''^*, \Sigma''^*, E'', \xi'', \nu'' \rangle \in F$,

$\xi'' \subseteq \xi$ and $\nu'' \leq \nu'$ by corresponding result for η;

and $\forall I$ and $\nu_1 > \eta'[\![\mathbf{gen}]\!]$,

$\nu_1 > \nu$;

so $\gamma'[\![\mathbf{fn}]\!][\![I]\!]\nu_1 = \gamma[\![\mathbf{fn}]\!][\![I]\!]\nu_1 = \langle \langle\rangle, \langle\rangle, \langle\rangle, \langle\rangle \rangle$;

or $\phi \lessdot \phi^*$ ie $\phi^* = \langle \phi \rangle$ and $I \lessdot \mathit{I}[\![\Phi]\!]$ ie $I = I'$;

hence $I^* = I'^*$ etc.

and $\nu = \nu'$;

so $\mathit{IdeL}[\![E]\!]\gamma\nu \leq \mathit{IdeL}[\![E]\!]\gamma'\nu \leq \mathit{IdeL}[\![E]\!]\gamma\nu \hat{} \mathit{IdeL}[\![E']\!]\gamma\nu \equiv \mathit{IdeL}[\![E]\!]\gamma\nu$

since $E = E'$;

ie $\mathit{IdeL}[\![E]\!]\gamma'\nu \equiv \mathit{IdeL}[\![E']\!]\gamma\nu \equiv \gamma'[\![\mathbf{fn}]\!][\![I]\!]\nu[\![\mathbf{lfree}]\!]$

by definition of η' ;

similarly for *RIdeR* ;

also $\Sigma^* = \Sigma'^* = \gamma'[\![\mathbf{fn}]\!][\![I]\!]\nu[\![\mathbf{spe}]\!]$;

and $I^* = I'^* = \gamma'[\![\mathbf{fn}]\!][\![I]\!]\nu[\![\mathbf{par}]\!]$;

and $\nu = \eta'[\![\mathbf{gen}]\!]$;

and $\xi = \eta'[\![\mathbf{var}]\!]$;

and $\forall I'', \iota''$ st $\xi[\![I'']\!]\downarrow\iota'' = \phi'' = \langle I''^*, \Sigma''^*, E'', \xi'', \nu'' \rangle$,

$\phi''=\eta'[\![\mathsf{var}]\!][\![I'']\!]\downarrow\iota''$;

so either $\phi''=\phi$

or $\phi''=\eta[\![\mathsf{var}]\!][\![I'']\!]\downarrow\iota_1''$;

in either case,

$\xi''\subseteq\eta'[\![\mathsf{var}]\!]=\xi$ and $\nu''\leq\eta'[\![\mathsf{gen}]\!]=\nu$;

and $\forall I$ and $\nu_1>\eta'[\![\mathsf{gen}]\!]$,

$\gamma'[\![\mathsf{fn}]\!][\![I]\!]\nu_1=\gamma[\![\mathsf{fn}]\!][\![I]\!]\nu_1=\langle\langle\rangle,\langle\rangle,\langle\rangle,\langle\rangle\rangle$;

Hence η' is Fn-Compatible.

QED

Lemma 4.4.3:

If η is Fn-Compatible

and $\phi=\langle I^*,\Sigma^*,E,\xi,\nu\rangle$ satisfies $\xi\subseteq\eta[\![\mathsf{var}]\!]$

and each $\delta_1\in L+V$;

then $\eta'=\mathit{NoteAppn}[\![E]\!](\eta[\xi[\delta^{*\wedge}/I^*]/\mathsf{var}][\lambda I.\mathit{Error}/\mathsf{cont}])$

is Fn-Compatible.

Proof

let I,ι be st $\eta'[\![\mathsf{var}]\!][\![I]\!]\downarrow\iota=\phi'=\langle I'^*,\Sigma'^*,E',\xi',\nu'\rangle\in F$;

then $\eta'[\![\mathsf{var}]\!]=\xi[\delta^{*\wedge}/I^*]$;

so either $\phi'=\xi[\![I]\!]\downarrow\iota_1$ for some ι_1 ;

in which case $\phi'=\eta[\![\mathsf{var}]\!][\![I]\!]\downarrow\iota_2$ for some ι_2 ;

so by Fn-Compatibility of η and since $\gamma=\gamma'$,

$\mathit{IdeL}[\![E']\!]\gamma'\nu'\equiv\gamma'[\![\mathsf{fn}]\!][\![I]\!]\nu'[\![\mathsf{lfree}]\!]$;

and similarly for *RIdeR* ;

also $\Sigma^*=\gamma'[\![\mathsf{fn}]\!][\![I]\!]\nu'[\![\mathsf{spe}]\!]$;

and $I^*=\gamma'[\![\mathsf{fn}]\!][\![I]\!]\nu'[\![\mathsf{par}]\!]$;

and $\nu'\leq\eta[\![\mathsf{gen}]\!]=\eta'[\![\mathsf{gen}]\!]$;

and $\xi'\subseteq\xi\subseteq\eta'[\![\mathsf{var}]\!]$;

and if $\xi'[\![I'']\!]\downarrow\iota''=\phi''$ then $\xi''\subseteq\xi'$;

or $\phi' \prec \delta^*$;

which is impossible since each $\delta_1 \in L+V$.

QED

The corresponding result to theorem 3.4 can now be proved. Again, only the cases that are different from the ELM version need be considered. In particular, one subtle change in the definitions of the functions *IdeL* and *IdeR* may be noted - the cases $[\![$ let Δ in $\Theta_1]\!]$ and $[\![$ let Δ in $E_1]\!]$ no longer remove the declared identifiers from the list obtained from Θ_1 and E_1. This is a rather unfortunate consequence of replacing the single denotations of the $[\![$ var $]\!]$ component of ELM environments by lists of denotations. It is a considerable set-back, since it prevents the compiler from recognising that variables declared local to a program section cannot possibly cause interference with other sections of the program. A possible resolution of this would be to ban from the language declarations which cause holes in scope, ie which redeclare previously declared identifiers; then the D^* form of environment would no longer be needed and the problem would not arise. POPLAR, however, has the D^* form, and so will require that identifiers used for local variables in parallel tasks are distinct. This is not an intolerable constraint, since a preprocessing phase could be conceived which would transform the identifiers of a program to avoid such clashes, while keeping essentially the same semantics. This would not work, however, in the case where different iterations of a loop are to be considered for parallel execution, as in the next chapter.

Theorem 4.4:

If the conditions of theorem 3.4 are satisfied

and also η is Fn-Compatible,

then any $\chi \triangleleft \mathit{Develop}(C[\![\Theta]\!]\eta\theta_t)\chi^*\sigma$ is Sanctioned by

$\eta, \mathit{IdeL}[\![\Theta]\!]\gamma\nu, \mathit{IdeR}[\![\Theta]\!]\gamma\nu$

where $\nu = \mathit{MaxGen}\eta \leq \eta[\![\mathtt{gen}]\!]$

and $\mathit{MaxGen}\eta = \mathit{Max}(\underset{\mathrm{I}\ \phi.\mathit{Denoted}\phi\mathrm{I}\eta}{\mathit{Max}}(\gamma[\![\mathtt{gen}]\!]))$;

and similarly for *E*, *L*, *R*, *A* and *D*.

Proof

Case $[\![\mathtt{let}\ \Phi\ \mathtt{in}\ E_1]\!]$:

let $\nu' = \eta[\![\mathtt{gen}]\!]$

and $\langle\phi\rangle = \mathit{Fix}(\lambda\phi^*.G[\![\Phi]\!](\eta[\phi^*\hat{}/\mathtt{var}/I[\![\Phi]\!]])(\nu'+1))$

and $\eta' = \mathit{NoteAbs}[\![\Phi]\!](\eta[\phi^*\hat{}/\mathtt{var}/I[\![\Phi]\!]])(\nu'+1)$

and let Φ be $[\![I[I^*;\Sigma^*] \equiv E']\!]$;

then by Fn-Compatibility, $\nu \leq \nu'$

and $E[\![E]\!]\eta\kappa_t = E[\![E_1]\!]\eta'\kappa_t$

and $\mathit{MaxGen}\eta' = \mathit{Max}(\mathit{MaxGen}\eta, \nu'+1) = \nu'+1$;

hence $\chi \triangleleft \mathit{Develop}(E[\![E]\!]\eta\kappa_t)\chi^*\sigma$ is Sanctioned by

$\eta', \mathit{IdeL}[\![E_1]\!]\gamma'(\nu'+1), \mathit{IdeR}[\![E_1]\!]\gamma'(\nu'+1)$;

but $\mathit{IdeL}[\![E_1]\!]\gamma'(\nu'+1) \leq \mathit{IdeL}[\![E_1]\!]\gamma(\nu'+1) \hat{}\ \mathit{IdeL}[\![E']\!]\gamma(\nu'+1)$ (4.4.2)

$\leq \mathit{IdeL}[\![E_1]\!]\gamma\nu' \hat{}\ \mathit{IdeL}[\![E']\!]\gamma\nu'$ (4.4.1)

$\leq \mathit{IdeL}[\![E]\!]\gamma\nu'$;

and similarly for *IdeR* ;

also, $\forall\alpha, I,\ \mathit{Denoted}\alpha\mathrm{I}\eta \Leftrightarrow \mathit{Denoted}\alpha\mathrm{I}\eta'$

since $\phi \notin L$;

and $\eta'[\![\mathtt{proc}]\!] = \eta[\![\mathtt{proc}]\!]$;

thus each $\chi \triangleleft \mathit{Develop}(E[\![E]\!]\eta\kappa_t)\chi^*\sigma$ is Sanctioned by

$\eta, \mathit{IdeL}[\![E]\!]\gamma\nu, \mathit{IdeR}[\![E]\!]\gamma\nu$.

Case $[\![E_1[E^*]]\!]$:

Again the induction on $\#(Develop(E[\![E]\!]\eta\kappa_t)\chi^*\sigma)$ needs to be invoked.

If $\chi \triangleleft Develop(E[\![E]\!]\eta\kappa_t)\chi^*\sigma$

then by Regularity, one of the following holds:

(1) $\chi \triangleleft Develop(A[\![E_1]\!]\eta\kappa_t)\chi^*\sigma$;

so χ is Sanctioned by $\eta, IdeL[\![E_1]\!]\gamma\nu, IdeR[\![E_1]\!]\gamma\nu$;

(2) χ is *SetError* ;

(3) $\chi \triangleleft Develop(SEval[\![\Sigma^*]\!][\![E^*]\!]\eta_1\kappa_t')\chi_1^*\sigma_1$

where $\Sigma^* = \phi[\![spec]\!]$

where ϕ satisfies $\tau_{5\eta}\phi$

and $\eta_1 = \eta[\pi_1/proc]$

and $\eta[\![proc]\!] < \pi_1$;

but by structural induction,

any $\chi' \triangleleft Develop(L[\![E_\iota]\!]\eta'\kappa_t)\chi^*\sigma$ is Sanctioned by

$\eta', IdeL[\![E_1]\!]\gamma\nu, IdeR[\![E_\iota]\!]\gamma\nu$;

and similarly for $R[\![E_\iota]\!]$;

so any $\chi' \triangleleft Develop(S[\![\Sigma_\iota]\!][\![E_\iota]\!]\eta'\kappa_t)\chi^*\sigma$ is Sanctioned by

$\eta', IdeL[\![E_\iota]\!]\gamma\nu, RIdeR[\![E_\iota]\!]\gamma\nu$;

hence, by 2.5, χ is Sanctioned by

$\eta_1, \bigwedge_\iota IdeL[\![E_\iota]\!]\gamma\nu, \bigwedge_\iota RIdeR[\![E_\iota]\!]\gamma\nu$;

and therefore by $\eta, IdeL[\![E]\!]\gamma\nu, IdeR[\![E]\!]\gamma\nu$;

(4) $\chi \triangleleft Develop(ApplyFn[\![E_1]\!]\eta_2\phi\delta^*\kappa_t)\chi_2^*\sigma_2$

where for each ι, $(\delta_\iota \in L \wedge \tau_{0\eta}\delta_\iota) \vee (\delta_\iota \in V)$

and $\eta_2 = \eta[\pi_2/proc]$

and $\eta[\![proc]\!] < \pi_2$;

ie $\chi = \lambda\sigma.\sigma$

or $\chi \triangleleft Develop(R[\![E']\!]\eta'\kappa_t)(\chi_2^* \dagger 1)\sigma_2$

where $\eta' = NoteAppn[\![E_1]\!](\eta_2[\xi[\delta^* \hat{} / I^*]/var][\lambda I.Error/cont])$;

hence χ is Sanctioned by $\eta', IdeL[\![E']\!]\gamma'\nu'', RIdeR[\![E']\!]\gamma'\nu''$

where $\nu'' = MaxGen\eta' \leq \nu'$;

but $\tau_{5\eta}\phi$;

so $\exists I \leftarrow AIdeVal[\![E_1]\!]$ st $Denoted\phi I\eta$

and η is Fn-Compatible;

so $IdeL[\![E']\!]\gamma'\nu'' \leq IdeL[\![E']\!]\gamma'\nu'$

$\leq \gamma'[\![fn]\!][\![I]\!]\nu'[\![lfree]\!]$

$\leq \gamma_2[\![fn]\!][\![I]\!]\nu'[\![lfree]\!]$

$\leq \gamma[\![fn]\!][\![I]\!]\nu'[\![lfree]\!]$

$\leq FreeLVal[\![E_1]\!]\gamma\nu'$

$\leq FreeLVal[\![E_1]\!]\gamma\nu$

$\leq IdeL[\![E]\!]\gamma\nu$;

and similarly $IdeR[\![E']\!]\gamma'\nu'' \leq IdeR[\![E]\!]\gamma\nu$;

also $\xi \subseteq \eta[\![var]\!]$;

so for any α, if $Denoted\alpha I\eta'$,

then either $\alpha = \xi[\![I]\!]'\iota_1$ for some ι_1

$= \eta[\![var]\!][\![I]\!]\downarrow\iota_2$ for some ι_2 ;

or $\alpha \leftarrow \delta^*$;

so if χ is $Put\alpha\beta$,

then either $Denoted\alpha I\eta'$

for some $I \leftarrow IdeL[\![E']\!]\gamma'\nu'' \leq IdeL[\![E]\!]\gamma\nu$;

in which case,

either $Denoted\alpha I\eta$ with $I \leftarrow IdeL[\![E]\!]\gamma\nu$

or $\alpha = \delta_\iota$ for some ι;

so either $\eta[\![proc]\!] \leq Owner\alpha$

or $\exists I \leftarrow IdeVal[\![E_\iota]\!]$ st $Denoted\alpha I\eta$;

or $\eta[\![proc]\!] \leq Owner\alpha$;

hence either $Denoted\alpha I\eta$ for some $I \leftarrow IdeL[\![E]\!]\gamma\nu$

or $\eta[\![proc]\!] \leq Owner\alpha$;

and similarly for $CheckFetch\alpha\beta$;

Thus χ is Sanctioned by $\eta, IdeL[\![E]\!]\gamma\nu, IdeR[\![E]\!]\gamma\nu$.

QED

4.5 The Syntactic Conditions for POPLAR

Having established that the semantic and syntactic equations for POPLAR satisfy essentially similar relationships to those of ELM, with the proviso of Fn-Compatibility of environments, the parts can now be assembled in the same sort of way. The main difference still to be encountered is that environments can no longer, of course, be expected to be Share-Free. Instead, the [share] component mentioned in 4.0 comes into play, and the weaker condition of Share-Compatibility is used. The P-Strict condition remains as in chapter 3, its definition being only slightly modified to take into account the different structure of H.

Definition

η is *Share-Compatible* if

$$\forall I_1, I_2, \iota_1, \iota_2,$$
$$\eta[\![var]\!][\![I_1]\!]\downarrow\iota_1 = \eta[\![var]\!][\![I_2]\!]\downarrow\iota_2 \Rightarrow$$
$$(I_1 = I_2 \vee I_1 \prec \eta[\![share]\!] \vee I_2 \prec \eta[\![share]\!]) .$$

Definition

η is *P-Strict* if

$$\forall I, \iota,$$
$$\eta[\![var]\!][\![I]\!]\downarrow\iota = \alpha \in L \Rightarrow Owner\alpha \gtrsim \eta[\![proc]\!] .$$

The functions *CSeparable* etc can now be defined as for ELM,

but allowing for this weaker condition.

Definition

$$CSeparable[\![\theta^*]\!]\gamma =$$

$$\forall \iota,\iota',(\iota=\iota' \vee ShareDisjoint(IdeL[\![\theta_\iota]\!]\gamma(\gamma[\![gen]\!]))$$

$$(IdeL[\![\theta_{\iota'}]\!]\gamma(\gamma[\![gen]\!]) \wedge IdeR[\![\theta_{\iota'}]\!]\gamma(\gamma[\![gen]\!]))\gamma)$$

where $ShareDisjoint I_1^* I_2^* \gamma =$

$$\sim(\exists I_1, I_2,\ I_1 \lessdot I_1^* \wedge I_2 \lessdot I_2^* \wedge (I_1 - I_2 \vee I_1 \lessdot \gamma[\![share]\!] \vee I_2 \lessdot \gamma[\![share]\!]))$$

Lemma 4.5.1:

as lemma 3.5.1 but with $CSeparable[\![\theta^*]\!]\gamma$

and η is P-Strict and Share-Compatible.

Proof

as 3.5.1 except for case 1a,

when $\alpha=\eta[\![var]\!][\![I]\!]\downarrow\iota$ for some $I \lessdot IdeL[\![\theta_\iota]\!]$ and some ι ;

and $\alpha=\eta[\![var]\!][\![I']\!]\downarrow\iota'$ where $I' \lessdot IdeL[\![\theta_\iota]\!] \wedge IdeR[\![\theta_\iota]\!]$;

so by Share-Compatibility,

$I_1=I_2$ or $I_1 \lessdot \eta[\![share]\!]$ or $I_2 \lessdot \eta[\![share]\!]$;

which is precisely contradicted by the condition

$CSeparable[\![\theta^*]\!]\gamma$.

QED

Useful P-Tests can now be established for POPLAR, as they were for ELM.

Lemma 4.5.2:

$(\lambda\Theta^*\eta.CSeparable[\![\Theta^*]\!]\gamma\wedge\forall\iota\leq\#\Theta^*,\#Jumps[\![\Theta_\iota]\!]=0)$ is a P-Test for C wrt ($\lambda\eta.\eta$ is Share-Compatible and P-Strict).

Proof

as 3.5.2.

QED

Theorem 4.5:

If η_s and η_p are Share-Compatible (instead of Share-Free), and also Fn-Compatible,
and otherwise the same conditions are satisfied as in 3.5,
then the conclusions of theorem 3.5 also hold,
together with similar conclusions about A and F.

Proof

By structural induction, mainly as in 3.5.

Case $[\![$ **let** Δ **in** $\Theta_1]\!]$:

The proof is as in 3.5, except that η_s'' and η_p'' must be shown to be Share-Compatible and Fn-Compatible rather than Share-Free, and the proof of P-Strictness requires minor modifications to deal with D^* instead of D.

let $\eta_s''=\eta_s[\varepsilon^*\hat{}/var/I[\![\Delta]\!]][\pi_2/proc]$;
for any I,ι, if $I\triangleleft I[\![\Delta]\!]$,
then if $\iota=1$ then $\eta_s''[\![var]\!][\![I]\!]\downarrow\iota\triangleleft\varepsilon^*$;
and if $\iota>1$ then $\eta_s''[\![var]\!][\![I]\!]\downarrow\iota=\eta_s[\![var]\!][\![I]\!]\downarrow(\iota-1)$;
otherwise $\eta_s''[\![var]\!][\![I]\!]\downarrow\iota-\eta_s[\![var]\!][\![I]\!]\downarrow\iota$;
and P-Strictness follows in each case as in 3.5;

consider any $I_1 \neq I_2, i_1, i_2$

st $\eta_s''[\![\mathbf{var}]\!][\![I_\iota]\!]\downarrow \iota_\iota = \alpha_\iota \in L$ for $\iota = 1,2$;

If $I_\iota \lessdot \mathit{I}[\![\Delta]\!]$ and $\iota_\iota = 1$ (1)

for both $\iota = 1,2$,

then α_1, α_2 are distinct elements of ε^*, so $\alpha_1 \neq \alpha_2$;

If (1) holds for $\iota = 1$ but not for $\iota = 2$ (or vice versa),

then $\alpha_1 \lessdot \varepsilon^*$ so $\pi < \pi_2 \leq \mathit{Owner}\alpha_1$;

but $\alpha_2 = \eta_s[\![\mathbf{var}]\!][\![I_2]\!]\downarrow \iota_2'$

where $\iota_2' = (I_2 \lessdot \mathit{I}[\![\Delta]\!] \rightarrow \iota_2 - 1, \iota_2)$;

so $\sim\pi \leq \mathit{Owner}\alpha_2$;

so $\alpha_1 \neq \alpha_2$;

If (1) holds for neither $\iota = 1$ or 2,

then $\alpha_\iota = \eta_s[\![\mathbf{var}]\!][\![I_\iota]\!]\downarrow \iota_\iota'$;

so by Share-Compatibility of η_s,

if $\alpha_1 = \alpha_2$

either $I_1 \lessdot \eta_s[\![\mathbf{share}]\!] = \eta_s''[\![\mathbf{share}]\!]$

or $I_2 \lessdot \eta_s[\![\mathbf{share}]\!] = \eta_s''[\![\mathbf{share}]\!]$;

Hence η_s'' is Share Compatible;

and similarly for $\eta_p'' = \eta_p[\varepsilon^*\hat{}/\mathbf{var}/\mathit{I}[\![\Delta]\!]][\pi_2/\mathbf{proc}]$;

Fn-Compatibility of η_s'' is obvious, since as $\varepsilon^* \in [L+V]^*$,

for any $\phi \in F$, $I \in Ide$,

$\mathit{Denoted}\phi I\eta_s \Leftrightarrow \mathit{Denoted}\phi I\eta_s''$.

Case $[\![\,\mathbf{let}\ \Phi\ \mathbf{in}\ \Theta_1\,]\!]$:

This is similar to the above case, except that the first two possibilities for $I_1, I_2, \iota_1, \iota_2$ cannnot occur since none of the denoted objects being "added" to the environment are locations,

and 4.4.2 gives the required Fn-Compatibility.

Case $[\![E_1[E^*]]\!]$:

This case proceeds similarly to the others, except that again the double induction (first on $\#Expand(E[\![E]\!]\eta\kappa_t)\sigma$) needs to be invoked.

As before, Regularity and structural induction imply that

either $Expand(E_s[\![E]\!]\eta_s\kappa_s)\sigma = Expand(A_s[\![E_1]\!]\eta_s\kappa_t)\sigma$

$\approx Expand(A_p[\![E_1]\!]\eta_p\kappa_t)\sigma$

$\approx Expand(E_p[\![E]\!]\eta_p\kappa_p)\sigma$

or $Expand(E_s[\![E]\!]\eta_s\kappa_s)\sigma = Expand(A_s[\![E_1]\!]\eta_s\kappa_t)\sigma\hat{}\langle SetError\rangle$

$\approx Expand(A_p[\![E_1]\!]\eta_p\kappa_t)\sigma\hat{}\langle SetError\rangle$

$\approx Expand(E_p[\![E]\!]\eta_p\kappa_p)\sigma$

or $Expand(E_s[\![E]\!]\eta_s\kappa_s)\sigma =$

$Expand(A_s[\![E_1]\!]\eta_s\kappa_t)\sigma\hat{}\langle\chi\rangle\hat{}$

$Expand(SEval_s[\![\Sigma^*]\!][\![E^*]\!]\eta_s'\{\lambda\delta^*.ApplyFn[\![E_1]\!]\eta_s''\phi\delta^*\kappa_s\})\sigma_1$

where $\chi = (CheckProcs\pi\langle\pi_1,\pi_2\rangle)$

and $\pi = \eta_s[\![\mathbf{proc}]\!]$

and $\pi < \pi_1, \pi_2$

and $\eta_s' = \eta_s[\pi_1/\mathbf{proc}]$

and $\eta_s'' = \eta_s[\pi_2/\mathbf{proc}]$

for some $\phi = \langle I^*, \Sigma^*, E', \xi, \nu\rangle$

st $\tau_5\eta_s\phi$ and $\#E^* = \#I^*$

ie $\exists I \triangleleft AIdeVal[\![E_1]\!]$ st $Denoted\phi I\eta_s$;

but either $Expand(SEval_s[\![\Sigma^*]\!][\![E^*]\!]\eta_s'\kappa_s')\sigma_1$

$= Expand(SEval_s[\![\Sigma^*]\!][\![E^*]\!]\eta_s'\kappa_t')\sigma_1$

$\approx Expand(SEval_p[\![\Sigma^*]\!][\![E^*]\!]\eta_p'\kappa_t')\sigma_1$

$\approx Expand(SEval_p[\![\Sigma^*]\!][\![E^*]\!]\eta_p'\kappa_p')\sigma_1$;

so that $Expand(E_s[\![E]\!]\eta_s'\kappa_s)\sigma \approx Expand(E_p[\![E]\!]\eta_p'\kappa_p)\sigma$;

or $Expand(SEval_s[\![\Sigma^*]\!][\![E^*]\!]\eta_s'\kappa_s')\sigma_1$

$= Expand(SEval_s[\![\Sigma^*]\!][\![E^*]\!]\eta_s'\kappa_t')\sigma_1 \hat{} Expand(\kappa_s'\delta^*)\sigma_2$

where δ^* satisfies

for $1 \le \iota \le \#\delta^*$

if $\Sigma_\iota = lv$ then $\tau_{2\eta_s'}\delta_\iota$;

if $\Sigma_\iota = rv$ then $\tau_{3\eta_s'}\delta_\iota$;

ie $Expand(E_s[\![E]\!]\eta_s\kappa_s)\sigma$

$= \chi_s^* \hat{} Expand(ApplyFn[\![E_1]\!]\eta_s''\phi\delta^*\kappa_s)\sigma_2$

$= \chi_s^* \hat{} (\lambda\sigma.\sigma) \hat{} Expand(R[\![E']\!]\eta'\kappa_s)\sigma_2$

where $\eta' = NoteAppn[\![E_1]\!]\eta''$

and $\eta'' = \eta_s''[\xi[\delta^* \hat{} /I^*]/var][\lambda I.Error/cont]$;

by Fn-Compatibility of η_s, $\xi \subseteq \eta_s[\![var]\!] = \eta_s''[\![var]\!]$;

and $\eta'[\![var]\!] = \eta''[\![var]\!]$;

so $Denoted\alpha I\eta' \Rightarrow (Denoted\alpha I\eta_s \vee \alpha \lessdot \delta^*)$;

so if $\alpha \in L$ and $Denoted\alpha I\eta'$,

then either $Denoted\alpha I\eta_s$;

so $Owner\alpha \gtrsim \pi$;

hence $Owner\alpha \gtrsim \pi_2 = \eta'[\![proc]\!]$;

or $\alpha \lessdot \delta^*$;

so $\tau_{2\eta_s'}\alpha$;

hence either $Denoted\alpha I'\eta_s'$

for some $I' \lessdot IdeVal[\![E_1]\!]$;

in which case the previous case again holds

or $\pi_1 = \eta_s'[\![proc]\!] \le Owner\alpha$;

so $\pi_2 \gtrsim Owner\alpha$;

hence η' is P-Strict;

Now consider any $I' \neq I''$ and α st

$Denoted\alpha I'\eta'$ and $Denoted\alpha I''\eta'$;

then either $Denoted\alpha I'\eta_s$ and $Denoted\alpha I''\eta_s$;

so by Share-Compatibility of η_s,

either $I' \prec \eta_s[\![\text{share}]\!] \leq \eta'[\![\text{share}]\!]$

or $I'' \prec \eta_s[\![\text{share}]\!] \leq \eta'[\![\text{share}]\!]$;

or for at least one of I', I'' - say I' -

$I' \prec I^*$ and $\alpha \prec \delta^*$;

specifically, $\exists \iota$ st $I' = I_\iota$, $\alpha = \delta_\iota$ and $\Sigma_\iota = \text{lv}$;

so $I' \prec \textit{LParams} I^* \Sigma^*$;

but by Fn-Compatibility of η_s, since $\textit{Denoted}\phi I \eta_s$

$\Sigma^* = \psi[\![\text{spe}]\!]$;

$I^* = \psi[\![\text{par}]\!]$;

and $\nu \leq \eta_s[\![\text{gen}]\!] = \eta''[\![\text{gen}]\!]$

where $\psi = \gamma_s[\![\text{fn}]\!][\![I]\!]\nu = \gamma''[\![\text{fn}]\!][\![I]\!]\nu$;

so $I' \prec \textit{ShareList}(\textit{AIdeVal}[\![E_1]\!])\gamma''$;

so $I' \prec \eta'[\![\text{share}]\!]$;

so η' is Share-Compatible;

η' is Fn-Compatible by 4.4.3;

Result now follows immediately from induction.

QED

The simple conditions of chapter 3 have now been extended to include procedures - the results are not very promising! They are so complicated that any compiler-writer might be excused for shirking the task of implementing them. More will be said about this in chapter 6. Meanwhile, the next chapter will consider different language extensions.

CHAPTER 5: A LANGUAGE WITH ARRAYS AND LOOPS

"Alice couldn't help pointing her finger at Tweedledum, and saying 'First Boy!' 'Next Boy!' said Alice, passing on to Tweedledee... The next moment they were dancing round in a ring. This seemed quite natural (she remembered afterwards)."

Through the Looking Glass: Chap 4.

The last of the three example languages will now be described. It is called APPLE (Array-Processing Programming Language Example). The important feature of this language concerns an area which promises to give some of the best results in terms of efficient usage of parallel computers - namely the processing of different iterations of a loop concurrently. Since consideration of the sort of conditions involved, as for example derived in the last two chapters, indicates that such a loop may not assign to any simple free variable, the only way in which a parallel loop can do anything useful is to contain references to objects which denote different locations in different loop iterations; in other words, arrays or more complicated data structures must inevitably also be included.

One effect of introducing arrays into POPLAR is to duplicate the sharing problem - indeed, the sharing of arrays is slightly worse than that of locations, since it can occur with parameters called by R-value as well as those called by L-value; furthermore, matters become even more confused if an array element is used as an actual reference parameter, since then "mixed" sharing may occur - the denotation of one identifier being the same as a component of the denotation of another.

To avoid repeating much of the complex work of chapter 4, therefore, APPLE has none of the features introduced by POPLAR,

but instead uses ELM as a basis.

The form of the results in this chapter follows much the same pattern as for chapters 3 and 4, and there will be very little that is new (except for trivial modifications of functionalities) in the proofs, which will therefore be generally even more skeletal than in the previous chapters.

5.0 Description of APPLE

The syntactic domains of APPLE are slightly extended versions of the ELM ones; each of **Com**, **Exp** and **Dec** has just one new category. **Com** has a new form of loop - the **for** loop - which is more suitable for parallel processing than the **while** loop, both because the control variable provides a means of guaranteeing that locations accessed in different iterations are distinct, and also because it can be guaranteed to have a finite number of iterations, and furthermore, a number known before the loop is entered. (This is not true eg of Algol **for** loops, but it is in APPLE, since the limits are evaluated only once and the "control variable" cannot be changed within the loop body.) **Dec** allows an identifier to be declared to denote a new array of arbitrarily many dimensions - the upper bounds for the subscripts are given by the list of expressions in the declaration, and for simplicity, the lower bounds are all taken to be 1. The new category in **Exp** allows an array element to be referenced. It is also necessary to define **Bas** a little more, and demand that it includes a domain **Nml** of numerals which represent integers; thus it will be assumed that $B[\![N]\!]$ is an integer. Also, **Ope** will be expected to contain at least the

arithmetic operators + and ×.

The major change to the semantic structure in APPLE is a new domain R of arrays. An array has two components - a list of positive integers giving the upper bound for each subscript, and a list of locations with length equal to the product of the upper bounds. The details of the mapping from a list of subscripts to the corresponding location are not important, and are left to the unspecified function $GetArrayLoc \in [R \to I^* \to L]$; the only relevant properties of this function are:

i) if $\rho \in R$ and ι^* is a valid subscript list,

ie if $InArrayRange\,\rho\,\iota^*$

where $InArrayRange\,\rho\,\iota^* =$

$$(\#\iota^* = \#\rho[\![\text{bounds}]\!]) \wedge (\forall \iota \leq \#\iota^*, 1 \leq \iota_\iota \leq \rho[\![\text{bounds}]\!] \downarrow \iota)$$

then $GetArrayLoc\,\rho\,\iota^* < \rho[\![\text{locs}]\!]$;

and ii) if ι_1^* and ι_2^* both satisfy $InArrayRange\,\rho\,\iota_\iota^*$

and $\iota_1^* \neq \iota_2^*$

then $GetArrayLoc\,\rho\,\iota_1^* \neq GetArrayLoc\,\rho\,\iota_2^*$.

It will be assumed without detailed proofs that two different arrays do not share any locations, and also that an array cannot share a location with a simple variable. (This can happen in a language which allows an identifier to be made to denote an already existing L-value, eg by parameter call by reference, but not in APPLE.)

New arrays are claimed with the aid of the function *NewLocs* which behaves very like *NewLoc* except that it takes an extra argument - a non-negative integer - and returns (if possible) a list of the appropriate number of locations. These locations are all different, and all have the same *Owner*, so the function *Owner* can naturally be extended to apply also to arrays.

As with functions in the last chapter, E and D are extended to include R as a summand, but V is not.

To cope with arrays in the syntactic conditions, a new "semi-syntactic" domain J is introduced, which replaces Ide as the domain from which results of *IdeL* etc are taken. Its use will be described in detail later.

The environment will not require all the mechanism of POPLAR's syntactic environments, but it does need one simple "syntactic" component: lit:[Ide→T] which serves to indicate whether an identifier is a constant (denotes an element of B) or a variable (denotes an element of L or R). Hence the following compatibility condition:

Definition

η is *Lit-Compatible* if

$\forall I \in Ide,\ \eta[\![lit]\!][\![I]\!] \Leftrightarrow \eta[\![var]\!][\![I]\!] \in V$.

The detailed semantics of APPLE appear in appendix 3. A new semantic function *Q* is used to deal with **for** loops; it has the same functionality as *C*, but with two extra parameters - the control variable I, and its value ι for the particular iteration concerned. A corresponding version of *Eval* - *QEval* is used which requires a P-Test with slightly different functionality - see section 5.5. The function *W* is partially defined corresponding to the partial specification of Ope.

5.1 Consistency of APPLE

The extension of theorem 3.1 to APPLE is extremely trivial - the result for Q follows immediately from that for C, and the proofs for the new syntactic categories follow exactly the same patterns as in 3.1.

Theorem 5.1:

If θ_0, κ_0, κ'_0, η are ν-Consistent
then $C[\![\Theta]\!]\eta\theta_0$, $Q[\![I]\!]\iota[\![\Theta]\!]\eta\theta_0$, $E[\![E]\!]\eta\kappa_0$, $L[\![E]\!]\eta\kappa_0$, $R[\![E]\!]\eta\kappa_0$
and $D[\![\Delta]\!]\eta\kappa'_0$ are ν-Consistent.

Proof

as for theorem 3.1.

QED

5.2 Termination in APPLE

The remarks of 5.1 apply equally to the results about termination, and for the same reasons, so no further comment is necessary.

Theorem 5.2:

Jumps$[\![\Theta]\!]$ is a T-List for $C[\![\Theta]\!]$ and $Q[\![I]\!]\iota[\![\Theta]\!]$;

Jumps$[\![E]\!]$ is a T-List for $E[\![E]\!]$, $L[\![E]\!]$ and $R[\![E]\!]$;

Jumps$[\![\Delta]\!]$ is a T-List for $D[\![\Delta]\!]$.

Proof

similar to 3.2.

QED

5.3 Regularity of APPLE

As intimated at the beginning of the chapter, the mechanism used for ELM and POPLAR of producing lists of identifiers representing possible values of an expression, and possible free variables of a program section, is inadequate to deal with arrays. In APPLE, therefore, more complicated lists are used, in which the components are elements of the domain J and can either be simple identifiers - denoting simple variables as previously - or elements of the domain $[Ide \times [I^* + \{?\}]^*]$. The latter form is used for array elements: the identifier denotes the array being accessed, and each component of the form $[I^* + \{?\}]$ represents one subscript. (ι^* will be used for elements of this sum, even in the case $\iota^* = ?$.) The representation is relative to a list of identifiers I^*: if the subscript is a linear expression in elements of I^* only, then ι^* is the corresponding list of coefficients, with the constant at the end; otherwise $\iota^* = ?$. The list I^* (all of whose components must denote constants) is given as a parameter to the functions *IdeVal*, *IdeL*, etc., and will include for loop control variables and constant free variables of a loop body. This interpretation of the elements of J referring to array elements is embodied in the following definition, which can then be combined with the simple variable case to give a complete interpretation of J in the definition of the APPLE form of *Denoted* (which now requires I^* as an extra argument).

Definition

$\alpha \in L$ is *Array-Denoted* by $\Psi=(I,\iota^{**})$ and I^* in η if

$\rho=\eta[\![\mathbf{var}]\!][\![I]\!] \in R$

and $InArrayRange\,\rho\,\iota^*$

and $\alpha=GetArrayLoc\,\rho\,\iota^*$

where ι^* satisfies:

$\#\iota^*=\#\iota^{**}$

and for $1\leq\iota\leq\#\iota^*$,

either $\iota^*_\iota=?$

or $\#\iota^*_\iota=\#I^*+1$ $(=\nu$ say)

$$\text{and } \iota_\iota=\sum_{\iota'=1}^{\nu-1}(\beta_\iota,\times\iota_{\iota\iota'},)+\iota_{\iota\nu}$$

where $\beta_\iota=\eta[\![\mathbf{var}]\!][\![I_\iota]\!]$.

$Denoted\alpha\Psi\eta I^*$ =

$(\Psi\in \mathbf{Ide} \wedge \varepsilon=\eta[\![\mathbf{var}]\!][\![\Psi]\!]) \vee$

$(\Psi\in \mathbf{J} \wedge \varepsilon$ is Array-Denoted by Ψ and I^* in $\eta)$.

The list I^* appears as a parameter in all the results for this and the next section with the constraint that all its components are different. (This constraint could be removed, but there is no particular reason to do so, and doing so forces extra complications on the equations.) There is one exception to this rule: in order to preserve the condition that each component denotes a constant, (the analysis becomes almost impossible if uncontrolled changes are allowed to variables in subscript expressions during the loop,) when a new declaration is encountered, I^* must be "purged" of any identifiers which are given non-constant denotations by the declaration. This cannot be done by simply omitting the offending identifiers if they occur in the list, since this would result in the mismatching of

coefficients at an outer level; instead they are replaced by a special dummy variable I_0 which is not allowed in programs and therefore causes no confusion if it is duplicated in I^*, since its coefficient will always be zero. This is summarised in the following:

Definition

DistinctI^* =

$$\#I^*=0 \vee ((I_1=I_0 \vee \sim I_1<(I^*\dagger 1)) \wedge \mathit{Distinct}(I^*\dagger 1)) \ .$$

The proof of Regularity is now straightforward, most of it being similar to 3.3. To deal with Array-Denotation, a lemma is first required to show that *IdeSub*, which is used by *IdeVal* to evaluate one subscript of an array element, behaves as expected. The main theorem then follows, only three cases being significantly different from previous work - the case that accesses an array element, and the two declaration cases which change the environment, in which the conditions on I^* must be shown still to hold.

Lemma 5.3.1:

If $\forall I<I^*$, $\eta[\![\ \mathsf{var}\ ]\!][\![I]\!] \in B$ (1)

and *Distinct*I^*

and $\iota^*=\mathit{IdeSub}[\![E]\!][\![I^*]\!] \neq ?$

then $\#\iota^*=\#I^*+1=\nu$ say,

and $R[\![E]\!]\eta$ is Regular wrt τ,τ' and

$$(\lambda\varepsilon.\ \varepsilon=\sum_{\iota=1}^{\#I^*}(\eta[\![\ \mathsf{var}\ ]\!][\![I_\iota]\!]\times\iota_\iota)+\iota_\nu) \qquad (2)$$

where $\tau=\mathit{Invariant}(\eta[\![\ \mathsf{proc}\ ]\!])$, $\tau'=\mathit{Subvariant}(\eta[\![\ \mathsf{proc}\ ]\!])$.

Proof

by structural induction and cases of E.

let ε_0 denote RHS of (2);

Case ⟦I⟧ :

$R⟦E⟧\eta\kappa = Rv\kappa(\eta⟦\text{var}⟧⟦I⟧)$

$= \kappa(\eta⟦\text{var}⟧⟦I⟧)$ because of (1);

but since $\iota^* \neq ?$, $I \triangleleft I^*$

and $\iota^* = \overset{\nu}{\underset{\iota=1}{\bigstar}} (I = I_\iota \rightarrow 1, 0)$;

so for $1 \leq \iota \leq \nu$, $\iota_\iota = (I = I_\iota \rightarrow 1, 0)$

and $\iota_\nu = 0$;

so $\varepsilon_0 = \sum_{\iota . I_\iota = I} (\eta⟦\text{var}⟧⟦I⟧)$;

but $I \triangleleft I^*$ and $Distinct I^*$ and I cannot be I_0 ;

so $I_\iota = I$ for precisely one ι ;

hence $\varepsilon_0 = \eta⟦\text{var}⟧⟦I⟧$;

so $R⟦E⟧\eta\kappa = \kappa\varepsilon_0$;

and Regularity follows immediately.

Case ⟦N⟧ :

$R⟦E⟧\eta\kappa = \kappa(B⟦N⟧)$;

and for $1 \leq \iota \leq \nu$, $\iota_\iota = 0$;

and $\iota_\nu = B⟦N⟧$;

so $\varepsilon_0 = B⟦N⟧$;

and again Regularity follows trivially.

Case ⟦I×N⟧ :

let $\beta = \eta⟦\text{var}⟧⟦I⟧ \in B$;

then by above two cases,

$REval(⟦I⟧, ⟦N⟧)\eta$ is Regular wrt τ, τ' and

$(\lambda\langle \varepsilon_1,\varepsilon_2\rangle .\varepsilon_1=\beta\wedge\varepsilon_2=B[\![N]\!])$;

so $R[\![I\times N]\!]\eta$ is Regular wrt $\tau,\tau',(\lambda\varepsilon.\varepsilon=W[\![\times]\!]\langle\beta,B[\![N]\!]\rangle)$;

ie wrt $\tau,\tau',(\lambda\varepsilon.\varepsilon=\beta\times B[\![N]\!])$;

but as in case $[\![I]\!]$,

$\iota_\iota=B[\![N]\!]$ for precisely one value of $\iota<\nu$,

and $\iota_\iota=0$ for all other values of ι ;

so $\varepsilon_0=\beta\times B[\![N]\!]$;

hence $R[\![I\times N]\!]\eta$ is Regular wrt $\tau,\tau',(\lambda\varepsilon.\varepsilon=\varepsilon_0)$.

Case $[\![N\times I]\!]$

is similar.

Case $[\![E_1+E_2]\!]$:

let $\iota'^*=IdeSub[\![E_1]\!][\![I^*]\!]$

and $\iota''^*=IdeSub[\![E_2]\!][\![I^*]\!]$;

so $\iota'^*\neq?\neq\iota''^*$

and $\#\iota^*=\#\iota'^*=\#\iota''^*$

and for $1\leq\iota\leq\#\iota^*$, $\iota_\iota=\iota'_\iota+\iota''_\iota$

and by induction,

$R[\![E_1]\!]\eta$ is Regular wrt $\tau,\tau',(\lambda\varepsilon.\varepsilon=\varepsilon'_0)$

where $\varepsilon'_0=\sum_{\iota=1}^{\#I^*}(\eta[\![var]\!][\![I_\iota]\!]\times\iota'_\iota)+\iota'_\nu$;

and similarly for $R[\![E_2]\!]\eta$;

hence $REval\langle[\![E_1]\!],[\![E_2]\!]\rangle\eta$ is Regular wrt

$\tau,\tau',(\lambda\langle\varepsilon_1,\varepsilon_2\rangle.\varepsilon_1=\varepsilon'_0\wedge\varepsilon_2=\varepsilon''_0)$;

so $R[\![E_1+E_2]\!]\eta$ is Regular wrt $\tau,\tau',(\lambda\varepsilon,\varepsilon=\varepsilon'_0+\varepsilon''_0)$;

but $\varepsilon_0'+\varepsilon_0'' = \sum_{\iota=1}^{\#I^*} (\eta[\![\text{var}]\!][\![I_\iota]\!]\times(\iota_\iota'+\iota_\iota'')) + (\iota_\nu'+\iota_\nu'')$

$= \varepsilon_0$.

QED

Theorem 5.3:

Exactly as theorem 3.3 except that

$\tau_{0\eta} = (\lambda\alpha.(\eta[\![\text{proc}]\!] \leq \mathit{Owner}\alpha) \vee$

$(\forall I^* \text{ st each } \eta[\![\text{var}]\!][\![I_\iota]\!] \in B \wedge \mathit{Distinct}I^*,$

$\exists \Psi \in \mathit{IdeVal}[\![E]\!][\![I^*]\!] \text{ st } \mathit{Denoted}\alpha\Psi\eta I^*))$.

Proof

The presence of the I^* parameter makes no difference except in the case $[\![\text{let } \Delta \text{ in } E_1]\!]$, because of the need to preserve the condition that each $\eta[\![\text{var}]\!][\![I_\iota]\!] \in B$.

The only other case which need be considered is the new one $[\![I{\downarrow}E^*]\!]$.

Case $[\![I{\downarrow}E^*]\!]$:

Consider any I^* st each $\eta[\![\text{var}]\!][\![I_\iota]\!] \in B$ and $\mathit{Distinct}I^*$;

then $\mathit{IdeVal}[\![E]\!]$ has precisely one component, viz $\Psi = (I, \iota^{**})$

where $\iota_\iota^* = \mathit{IdeSub}[\![E_\iota]\!][\![I^*]\!]$ for $1 \leq \iota \leq \#E^*$

and $\#\iota^{**} = \#E^*$;

so for each ι,

if $\iota_\iota^* \neq ?$ then by lemma 5.3.1,

$R[\![E_\iota]\!]\eta$ is Regular wrt $\tau,\tau',(\varepsilon.\varepsilon = \sum_{\iota'}^{\#I^*} (\beta_{\iota'} \times \iota_{\iota\iota'}) + \iota_{\iota\nu})$;

and if $\iota_\iota^* = ?$ then by induction,

$R[\![E_\iota]\!]\eta$ is Regular wrt $\tau,\tau',(\lambda\varepsilon.\mathit{True})$;

so $R[\![E_1]\!]\eta$ is Regular wrt τ,τ',

$$(\lambda\varepsilon.\iota_1^*=? \vee \varepsilon=\sum_{\iota'}^{\#I^*}(\beta_\iota,\times\iota_{\iota\iota'},)+\iota_{\iota\nu}) \ ;$$

so $REval[\![E^*]\!]\eta$ is Regular wrt

$$\tau,\tau',(\lambda\varepsilon^*.\forall\iota,(\iota_\iota^*=? \vee \varepsilon_\iota=\sum_{\iota'}^{\#I^*}(\beta_\iota,\times\iota_{\iota\iota'},)+\iota_{\iota\nu})) \ ;$$

hence $E[\![I{\downarrow}E^*]\!]\eta$ is Regular wrt τ,τ',

$$(\lambda\alpha.\exists\varepsilon^*,\#\varepsilon^*=\#E^*\wedge\forall\iota,(\iota_\iota^*=?\vee(\#\iota_\iota^*=\#I^*+1\wedge\varepsilon_\iota=\sum_{\iota'}(\beta_\iota,\times\iota_{\iota\iota'},)+\iota_{\iota\nu}))$$

$$\wedge\alpha=GetArrayLoc(\eta[\![\mathbf{var}]\!][\![I]\!])\varepsilon^*) \ ;$$

ie wrt τ,τ',($\lambda\alpha.\alpha$ is Array-Denoted by Ψ and I^* in η) ;

hence $E[\![I{\downarrow}E^*]\!]\eta$ is Regular wrt τ,τ',$(\lambda\varepsilon.\varepsilon\epsilon L\wedge\tau_{0\eta}\varepsilon)$.

Case [[**let** Δ **in** E_1]] :

let $I'^*=I[\![\Delta]\!]$;

It is sufficient to observe that if ε_0^* is st

$D[\![\Delta]\!]\eta$ is Regular wrt τ,τ',$(\lambda\varepsilon^*.\varepsilon^*=\varepsilon_0^*)$

(and if the first alternative of Regularity can ever occur),

then for each ι,

$I'_\iota{\leftarrow}Purge[\![I^*]\!][\![\Delta]\!] \Rightarrow \varepsilon_{0\iota}\epsilon B$;

for if Δ is [[I = E]] or [[$I \equiv$ **array** E^*]]

then $\varepsilon_0^*=(\varepsilon)$ where $\varepsilon\epsilon L+R$;

but $Purge[\![I^*]\!][\![\Delta]\!]=Nullify[\![I^*]\!][\![I]\!]$;

so $\sim I{\leftarrow}Purge[\![I^*]\!][\![\Delta]\!]$;

and if Δ is [[$I \equiv E$]]

then $\varepsilon_0^*=(\varepsilon)$ where $\varepsilon\epsilon B$;

and if Δ is [[Δ_1 **and** Δ_2]]

then result follows from induction;

hence if I^* satisfies each $\eta[\![\mathbf{var}]\!][\![I_\iota]\!] \in B$

then $I''^* = Purge[\![I^*]\!][\![\Delta]\!]$ satisfies each $\eta'[\![\mathbf{var}]\!][\![I''_\iota]\!] \in B$

where $\eta' = \eta[\varepsilon_0^*/\mathbf{var}/I[\![\Delta]\!]][NewLit[\![\Delta]\!]/\mathbf{lit}/I[\![\Delta]\!]]$;

and result follows inductively.

QED

5.4 Operations Produced by APPLE

The only new complication introduced by APPLE at this stage is the necessity to have I^* as an extra parameter in most of the functions. This means that the property Sanctioned is replaced by I^*-*Sanctioned* which is defined in exactly the same way except that I^* is appended to the arguments of *Denoted*, and that *IdeL*, *IdeR*, etc have an extra parameter.

Theorem 5.4:

For any $\eta, \chi^*, \sigma, I^*$ satisfying the same conditions as in 3.4

and also each $\eta[\![\mathbf{var}]\!][\![I_\iota]\!] \in B$ and $Distinct I^*$

then any $\chi \Leftarrow Develop(C[\![\theta]\!]\eta\theta_t)\chi^*\sigma$ is Sanctioned by

$\eta, IdeL[\![\theta]\!][\![I^*]\!], IdeR[\![\theta]\!][\![I^*]\!]$;

and similarly for the other semantic functions.

Proof

follows from 5.3 in exactly the same way as 3.4,

except for the extra parameter I^* in all the functions.

QED

5.5 The Syntactic Conditions for APPLE

Ananlogous results to those for ELM and POPLAR have now been proved for APPLE in the previous sections, but they involved the parameter I^* which could be arbitrary (subject to the constraints of Distinctness and constant denotations). This parameter must now be chosen so that the resulting conditions are useful. Since the aim is to show that (in suitable cases) the locations accessed in different iterations of a loop are different, the one thing that is guaranteed to be different, namely the control variable, is clearly an essential component. It is, in fact, advisable to make the list as long as possible in order to acquire complete information about as many array references as possible; certainly all the free variables (a misnomer here!) which are constant should be included, as these can be guaranteed to have the same value for each array reference.

This leads to the definition of *QSeparable*, which is intended to provide the Separability part of a P-Test for **for** loop evaluation. It uses a list I^* whose first component is the control variable of the loop, and it depends on the function *NonCoincident* which ensures that, for two given array element references in different loop iterations, there is at least one subscript which has a different value in the two references. No definition is given at this stage of *NonCoincident*, but only conditions sufficient to give *QSeparable* its required properties. The satisfaction of these conditions is the subject of the next section.

Definition

QSeparable⟦I⟧⟦Θ⟧ =

$\sim\exists I' \lessdot$ *IdeL*⟦Θ⟧⟨⟩ ∧

$(\forall \langle I', \iota_1^{**}\rangle \lessdot$ *IdeL*⟦Θ⟧⟦I^*⟧,

$\forall \langle I'', \iota_2^{**}\rangle \lessdot$ *IdeL*⟦Θ⟧⟦I^*⟧ ^ *IdeR*⟦Θ⟧⟦I^*⟧, st $I' = I'' \wedge \#\iota_1^{**} = \#\iota_2^{**}$,

NonCoincident $\iota_1^{**}\ \iota_2^{**}$)

where $I^* = \langle I\rangle$ ^ (*Literal*(*IdeR*⟦Θ⟧⟨⟩)(η⟦lit⟧)\⟨I⟩)

and *Literal* $I^*\gamma = (I^* = \langle\rangle \rightarrow \langle\rangle, (\gamma\llbracket I_1\rrbracket \rightarrow \langle I_1\rangle, \langle\rangle)$ ^ *Literal*$(I^* \dagger 1)\gamma$)

where *NonCoincident* satisfies

NonCoincident $\iota_1^{**}\ \iota_2^{**} \Rightarrow$

$\exists \iota \leq \#\iota_1^{**}$ st $\iota_{1\iota}^* \neq ? \neq \iota_{2\iota}^*$

and $\forall \iota_1', \iota_2', \iota^*$ st $\iota_1' \neq \iota_2' \wedge \#\iota^* = \#\iota_{1\iota}^* - 1 = \nu - 1$,

$$\iota_\iota' \times \iota_{1\iota 1} - \iota_2' \times \iota_{2\iota 1} + \sum_{\iota''=2}^{\nu-1} (\iota_{1\iota\iota''} - \iota_{2\iota\iota''}) \times \iota_{\iota''} + (\iota_{1\iota\nu} - \iota_{2\iota\nu}) \neq 0 \ .$$

The other forms of *Separable* are similarly redefined, but without any references to a control variable.

Since APPLE allows no sharing, the property of Share-Freeness is reinstated - although slightly modified, as is P-Strict, to allow for elements of R as well as those of L:

Definition

η is *Share-Free* if

$\forall I_1, I_2,\ \eta\llbracket \text{var}\rrbracket\llbracket I_1\rrbracket = \eta\llbracket \text{var}\rrbracket\llbracket I_2\rrbracket \in L+R \Rightarrow I_1 = I_2$.

Definition

η is *P-Strict* if

$$\forall I,\ \eta[\![\mathsf{var}]\!][\![I]\!] = \varepsilon \in L+R \Rightarrow Owner\varepsilon \gtrdot \eta[\![\mathsf{proc}]\!] \ .$$

Also needing slight modification is the definition of P-Test which will now accept a function taking a single command as parameter, instead of a list.

The process of putting together the entire conditions can now be completed as in previous chapters - the details are only given for the new and most complicated case - evaluating for loops with *QEval*.

Lemma 5.5.1:

If $QSeparable[\![I]\!][\![\Theta]\!]$

and $\theta^{\pi^*} = \langle \pi^*, \lambda\pi_1 . Q[\![I]\!]\iota[\![\Theta]\!]\eta[\pi_1/\mathsf{proc}]\theta_t \rangle$

and $\forall \pi_1 \lessdot \pi^*,\ \eta[\![\mathsf{proc}]\!] < \pi_1$

and η is ShareFree, P-Strict and Lit-Compatible,

then the conclusion of 3.5.1 holds.

Proof

let I^* be as in the definition of *QSeparable* ;

then certainly, $\forall I_1 \lessdot I^*,\ \eta[\![\mathsf{lit}]\!][\![I_1]\!] = True$;

and so since η is Lit-Compatible,

$$\eta[\![\mathsf{var}]\!][\![I_1]\!] \in V \ ;$$

hence the proof follows that of 3.5.1 with the parameter I^* added where appropriate, except for case 1a;

in which case $Denoted\alpha\Psi\eta_1$ for $\Psi \lessdot IdeL[\![\Theta]\!][\![I^*]\!]$

and $Denoted\alpha\Psi'\eta_1$, for $\Psi' \lessdot IdeL[\![\Theta]\!][\![I^*]\!] \hat{}\ IdeR[\![\Theta]\!][\![I^*]\!]$

where $\eta_\iota = \eta[\iota/var/I][True/lit/I]$;

Inspection of the definition of *IdeL* shows that whether or not an element of Ide is a component of $IdeL[\![\Theta]\!][\![I^*]\!]$ is independent of I^* ;

hence $\Psi \notin Ide$;

so since locations cannot share with components of arrays, $\Psi' \notin Ide$ also;

hence $\Psi = \langle I'', \iota^{**} \rangle$ and $\Psi' = \langle I', \iota'^{**} \rangle$;

so since different arrays cannot share elements,

$\eta_\iota[\![var]\!][\![I'']\!] = \eta_\iota,[\![var]\!][\![I']\!] = \rho \in R$;

so $I'' \neq I \neq I'$;

so $\eta[\![var]\!][\![I'']\!] = \eta[\![var]\!][\![I']\!] = \rho$;

ie by ShareFreeness, $I'' = I'$

and also $\#\iota^{**} = \#\rho[\![bounds]\!] = \#\iota'^{**}$;

so by QSeparability, $NonCoincident\ \iota^{**}\ \iota'^{**}$;

let ι'' be as in definition of $NonCoincident\ \iota^{**}\ \iota'^{**}$

and ι^*, ι'^* as in the definition of Array-Denoted

for ι^{**} and ι'^{**} respectively;

then $\iota^*_{\iota''} \neq ? \neq \iota'^*_{\iota''}$;

so $\iota_{\iota''} = \sum_{\iota'=1}^{\nu-1} (\beta_{\iota'} \times \iota_{\iota'' \iota'}) + \iota_{\iota'' \nu}$

where $\beta_\iota = \eta_\iota[\![var]\!][\![I_\iota]\!]$

and $\nu = \#I^* + 1$;

but $I_1 = I$ by definition of I^* ;

and $I_\iota \neq I$ for $\iota > 1$;

so $\beta_1 = \iota$;

and for $\iota > 1$, $\beta_\iota = \eta[\![var]\!][\![I_\iota]\!]$;

so $\iota_{\iota''} = \iota \times \iota_{\iota'' 1} + \sum_{\iota'=2}^{\nu-1} (\beta_{\iota'} \times \iota_{\iota'' \iota'}) + \iota_{\iota'' \nu}$;

and similarly $\iota'_{\iota''}=\iota'\times\iota'_{\iota''_1}+\sum_{\iota'=2}^{\nu-1}(\beta_{\iota'}\times\iota'_{\iota''_{\iota'}})+\iota'_{\iota''_\nu}$;

so $\iota_{\iota''}-\iota'_{\iota''}\neq 0$ by NonCoincidence since $\iota\neq\iota'$;

which is a contradiction since

$GetArrayLoc\rho\iota^*=\alpha=GetArrayLoc\rho\iota'^*$.

QED

Lemma 5.5.2:

For any I,

$(\lambda\Theta\eta.QSeparable[\![I]\!][\![\Theta]\!]\wedge\#Jumps[\![\Theta]\!]=0)$

is a P-Test for $\{Q[\![I]\!]\iota\}$

wrt ($\lambda\eta.\eta$ is ShareFree, P-Strict and Lit-Compatible).

Proof

as for lemma 3.5.2.

QED

Theorem 5.5:

as theorem 3.5.

Proof

as for theorem 3.5.

QED

5.6 Algorithms for Satisfying the Conditions

There was an important omission from the last section, which reveals where much of the complexity of dealing with arrays and loops lies, namely the function *NonCoincident* was not defined, but only certain conditions were imposed on it. To complete the picture, this section will discuss algorithms for satisfying

these conditions.

The conditions, as in 5.5, are applied to the subscript lists of two array element occurrences in different iterations of a loop. To be suitable for consideration, a subscript has to be a linear combination of the members of a certain set of variables; this set includes the control variable of the loop, which is certain to have a different value for the two occurrences, and free variables of the loop which remain constant for the duration of the loop; also, there is no particular reason to exclude other variables which may change their values during the loop. Non-linear subscripts could possibly be considered in certain cases, but the complexity of analysis and the amount of computation required, together with the observation (which should be taken as a reasonable hypothesis rather than a proved fact) that such expressions rarely occur, suggest that it is not worth including them.

Thus the subscripts to be considered will all take the form:

$$\sum_{\iota}\alpha_{\iota\kappa}X_{\iota}+\beta_{\kappa}Y+\sum_{\iota}\gamma_{\iota\kappa}Z_{\iota}+\delta_{\kappa}$$

where $\{X_{\iota}\}$ are fixed free variables

and Y is the control variable

and $\{Z_{\iota}\}$ are other variables, including the control variables of nested loops.

The corresponding κth subscript of the other array element will be:

$$\sum_{\iota}\alpha'_{\iota\kappa}X_{\iota}+\beta'_{\kappa}Y'+\sum_{\iota}\gamma'_{\iota\kappa}Z'_{\iota}+\delta'_{\kappa}$$

since the X_{ι} are the same.

The conditions require that there is some κ such that

$$\sum_{\iota}(\alpha_{\iota\kappa}-\alpha'_{\iota\kappa})X_{\iota}+(\beta_{\kappa}Y-\beta'_{\kappa}Y')+\sum_{\iota}(\gamma_{\iota\kappa}Z_{\iota}-\gamma'_{\iota\kappa}Z'_{\iota})+(\delta_{\kappa}-\delta'_{\kappa})\neq 0 \qquad (1)$$

for any $X_{\iota},Y,Y',Z_{\iota},Z'_{\iota}$ such that $Y\neq Y'$.

This can be considerably simplified by demanding (as is often the case) that all the corresponding coefficients (though not necessarily the constant term) are equal. Then (1) simplifies to:

$$\beta_{\kappa}Y''+\sum_{\iota}\gamma_{\iota\kappa}Z''_{\iota}+(\delta_{\kappa}-\delta'_{\kappa})\neq 0 \qquad (2)$$

for any $Y''\neq 0$ and any Z''_{ι} .

General methods for satisfying (1) or even (2) involve complex algebraic techniques which are beyond the scope of the present work. However, a few simple steps can be enumerated which will be effective in many actual situations.

1) In many cases there will be some κ for which $\beta_{\kappa}\neq 0$ but each $\gamma_{\iota\kappa}=0$ and $\delta_{\kappa}=\delta'_{\kappa}$, so (2) is clearly satisfied.

2) On the assumption that only integer variables are permitted, the condition that $(\delta_{\kappa}-\delta'_{\kappa})$ is not divisible by $HCF(\beta_{\kappa},\{\gamma_{\iota\kappa}\})$ also ensures that (2) is satisfied.

3) If neither of these works, eliminate some variables between the expressions for different κ, and try again.

These should suffice in most practical cases. Indeed, it may well be that far simpler conditions are enough.

Parallelism conditions have now been established for the third and last of the languages to be discussed. As with the procedure language, the conditions are fairly complicated; on the other hand, the gain to be expected from using them is much larger, as loops with many iterations can be completed in

roughly the time needed to execute the body once.

CHAPTER 6: CONCLUSION

"'I have answered three questions, and that is enough,'
Said his father, 'Don't give yourself airs!
Do you think I can listen all day to such stuff?
Be off, or I'll kick you downstairs!'"

Alice's Adventures in Wonderland: Chap 5.

The most striking general point about the results of the last three chapters must be their extreme complexity, especially when any but the most trivial language constructs are involved; indeed, several facilities, notably the possibility of storing anything except "basic" values, have not been included either because they make the conditions for parallelism prohibitively complicated or because no positive syntactic conditions are in general possible when they are permitted.

It is worth reviewing some of the main areas where difficulties lie, and considering some practical aspects of language design which could ameliorate the situation.

1) Sharing is a feature which can occur in most existing languages, though it probably does not occur in most programs - which is just as well for many who have done work in this field, since they almost all ignore it, giving it at most a passing mention. But this does suggest that, while sharing itself is not a particularly useful feature for the majority of programs, (indeed, its use is liable to make programs more obscure,) eliminating it completely from any language of reasonable power is rather difficult. Constructions which explicitly introduce sharing - eg a declaration by L-value as in [23] or Fortran's EQUIVALENCE (which, especially if combined with COMMON, can be

quite horrific) - can, of course, easily be dispensed with (or not considered in the first place). "Freezing" of free variables either at the declaration of, or on entry to, a procedure has been mooted in contexts other than parallel processing, and it goes a long way to solving the problem, at least in the way it appeared in chapter 4. But this still leaves the possibility of clashes between parameters called by reference (or worse, by name - imagine trying to subject Jensen's device to this sort of analysis), and without resorting to extremely arbitrary restrictions on the use of reference parameters, or abolishing completely a useful capability, there is no obvious way of eliminating it.

2) The ability to redclare an identifier, thus causing a hole in its scope, was seen in chapter 4 to cause complications. As with sharing, this does not seem to be a very useful feature, and again, it probably causes more confusion than anything else. Moreover, it is quite easily prohibited.

3) Procedures, as well as often introducing or aggravating the sharing problem, cause trouble inherently as was seen in chapter 4, since they are likely to change variables which do not appear explicitly in their calls. However, one of the techniques mentioned above - namely the freezing of free variables - also solves this problem, at least to the extent that information about procedures does not need to be kept by the compiler, although it probably has to be assumed (for want of further information) that all the actual parameters may be called by reference. If this is unsatisfactory, or if one is constrained to working with an existing language which doesn't do this, then the other easy method is never to consider sections of program containing procedure calls for parallel

processing. Unfortunately, this has the grave disadvantage of putting a high premium on the minimal use of procedures.

4) Various features available in some of the more sophisticated current languages wreak havoc with the detection of parallelism. Pointer variables, whereby locations can themselves be stored, constitute one such, since it is generally impossible to keep track, at compile time, of anything which is stored. Storing procedures, and also the ability to pass a procedure out of the scope of its free variables, as for example may happen if functions can return procedures as results, turn the sort of analysis which was required for POPLAR from a difficult exercise into a virtually impossible one; although they matter less if the approaches described in (3) are used.

5) All the conditions described so far have been concerned with the possibility of different processes accessing the same storage location. Another area where indeterminacy may arise is that of input and output; these impose constraints similar in many ways to those relating to the store, but the question of I/O has been fairly universally ignored, both here and in other work. This may be reasonable when considering the sort of program which does a small amount of input at the beginning, a small amount of output at the end, and a large amount of computation in the middle; but not for programs which do continual input or output; indeed, in such cases, totally different techniques and languages may well be more appropriate, especially when for example similar computations have to be performed on many different pieces of data, with more or less communication between them.

If I/O is ignored, the problem lies essentially in

synchronisation when more than one assignment is made to one variable in a program. An interesting and apparently rather drastic way of attacking this problem at the root involves the so-called "Single Assignment" languages, investigated in [10] and [33], in which a program may contain only one assignment to any one variable. Since this means the assignment statement becomes in effect an initialising declaration, it is perhaps more sensible to think of these as "Assignment-free" languages. Then since any variable (now a misnomer) cannot change its value after its initial declaration, any section of program may with impunity be allowed to start execution as soon as all its free variables have been declared. (This has affinities with Data Flow architectures [2], [11], [16] and such languages may well be appropriate for programming such machines, and conversely, Data Flow machines may be most suitable for implementing assignment-free languages.)

Of course, one cannot get very far by starting from a language like Fortran or Algol and simply removing the assignment statement (or replacing it by initialising declarations). Two major areas which would cause problems are assignments to free variables of loops and complex branching control structures, and assignments to components of data structures. The first requires some new constructions in the language, such as iteration operators which would correspond roughly to the summation, multiple product, and quantifying operators of conventional mathematics and others such as "the least value (in a given range) such that..." and even "the unique value such that...". Indeed, these operators, together with the sort of generalised switches suggested by [12] and [35], could be useful in multiple-assignment languages which are

destined for parallel analysis, especially if the expressions involved could be guaranteed to have no side effects. It is worth noting that a loop whose body fails only the JumpFree condition can often be reformulated as a "least value" loop followed by a JumpFree loop, both of which can be executed in parallel. Data structures provide a more difficult problem, one way round which would be to make a modified copy of a structure rather than assigning to a component, but this would inevitably be regarded as intolerably inefficient in terms of time or storage space (or both).

Some interesting points about programming techniques arise, even within the confines of a given language. Many of the features consequent on the concept of "Structured Programming" appear to make inherent parallelism easier to detect; eg the use of regular control structures like **if**, **while** and **for** instead of unbridled **goto**s makes isolation of possible subtasks easier; and proper block structuring, with variables which are only needed local to a block being declared as such, makes detection of independence far simpler. (cf [8] and [30] where a lot of work is done attempting to detect the use of the same variable for independent purposes in different blocks - ie overcoming poor programming practice.)

Taking this a stage further, it seems that the "higher the level" of programming language, the better. This should not be surprising, since a high level language requires the programmer to make fewer decisions about implementation details, and such details are just those which are drastically different when one is confronted with a parallel computer. (It is interesting to note in passing that [29], which explores parallelism exposure

techniques in Algol, claims that the overheads of the parallel analysis to be insignificantly small; while [13] admits that the work on exposing parallelism in Fortran requires vastly increased computation times, to the extent that a simple ad hoc heuristic algorithm has been devised to decide whether it is worth bothering to analyse a particular program for parallelism at all, in terms of the expected speed-up.)

There is a limit to how far this can be taken, however. As [32] points out, for many problems there are some algorithms which work better on parallel machines, and completely different ones which are better on sequential machines; but the day when a programmer's job will be to specify an abstract problem rather than a specific algorithm for its solution, so the machine can select its own algorithm to make best use of its resources, is far in the future, if indeed it ever comes. Nevertheless, a higher level of problem specification is undoubtedly a useful goal, and it would make the distinction between the three approaches described in chapter 0 rather less relevant.

In conclusion then, rigorous methods of programming language semantics have been applied to a practical problem of exposing potential parallelism in non-parallel languages. The results, unfortunately, are complicated and not encouraging for the styles of language which exist and are being designed today.

There is a need for languages designed with the claims of parallelism in mind. The trend must be towards making fewer assumptions about the nature of the machine involved, and the use of higher level concepts rather than the ad hoc addition of low level, explicitly parallel constructs to languages never designed for any but sequential machines; but as long as

languages anything like those commonly used today are used on highly parallel computers, the sorts of considerations described above will be virtually indispensable.

Appendix 1: Definition of ELM

"Its chief merit is its Simplicity - a Simplicity so pure, so profound, in a word, so *simple*, that no other word will fitly describe it. The meagre outline, and baldness of detail, of the present Chapter, are adopted in humble imitation of this great feature."

The New Belfry of Christ Church, Oxford.

Syntactic Domains:

$B \in$ Bas

$\Theta \in$ Com ::= $E_1 := E_2$

§I Θ^* §I

while E do Θ_1

if E do Θ_1

escape I

let Δ in Θ_1

$\Delta \in$ Dec ::= I = E

I ≡ E

Δ_1 and Δ_2

$E \in$ Exp ::= B

I

E_1 Ω E_2

Θ before E_1

$E_1 \rightarrow E_2, E_3$

let Δ in E_1

$I \in$ Ide

$\Omega \in$ Ope

Semantic Domains:

$\beta \in B$

$\theta \in C = [S \to [[S \to S] \times C]] + \{\theta_t\}$

$\delta \in D = L + B$

$\varepsilon \in E = L + B$

$\eta \in H = proc:P \times var:[Ide \to D] \times cont:[Ide \to C]$

$\iota \in I$

$\kappa \in K = E \to C$

$\kappa' \in K' = E^* \to C$

$\kappa'' \in K'' = E^{**} \to C$

$\alpha \in L$

$\nu \in N$

$\pi \in P$

$\sigma \in S = store:[L \to V] \times area:[L \to T] \times res:[P \to E] \times use:[P \to T] \times err:T$

$\tau \in T$

$V = B$

$\zeta \in Z = P^{\infty}$

$\chi \in [S \to S]$

Semantics:

$$C[\![E_1 := E_2]\!]\eta\theta =$$
$$\quad LREval(\langle [\![E_1]\!], [\![E_2]\!]\rangle)\,\eta(\lambda\langle\alpha,\beta\rangle.Update\,\alpha\beta\theta)$$

$$C[\![\S I\ \Theta^*\ \S I]\!]\eta\theta =$$
$$\quad CEval[\![\Theta^*]\!](\eta[\theta/\mathbf{cont}/I])\theta$$

$$C[\![\mathbf{while}\ E\ \mathbf{do}\ \Theta_1]\!]\eta\theta =$$
$$\quad Fix(\lambda\theta'.R[\![E]\!]\eta\{\lambda\varepsilon.\varepsilon \rightarrow C[\![\Theta_1]\!]\eta(Break\theta'),\ \theta\})$$

$$C[\![\mathbf{if}\ E\ \mathbf{do}\ \Theta_1]\!]\eta\theta =$$
$$\quad R[\![E]\!]\eta\{\lambda\varepsilon.\varepsilon \rightarrow C[\![\Theta_1]\!]\eta\theta,\ \theta\}$$

$$C[\![\mathbf{escape}\ I]\!]\eta\theta -$$
$$\quad \eta[\![\mathbf{cont}]\!][\![I]\!]$$

$$C[\![\mathbf{let}\ \Delta\ \mathbf{in}\ \Theta_1]\!]\eta\theta =$$
$$\quad ClaimProcs(\eta[\![\mathbf{proc}]\!])2\{\lambda\pi^*.D[\![\Delta]\!]\eta[\pi_1/\mathbf{proc}]$$
$$\qquad \{\lambda\varepsilon^*.C[\![\Theta_1]\!](\eta[\varepsilon^*/\mathbf{var}/I[\![\Delta]\!]][\pi_2/\mathbf{proc}])\theta\}\}$$

$$E[\![B]\!]\eta\kappa =$$
$$\quad \kappa(B[\![B]\!])$$

$$E[\![I]\!]\eta\kappa =$$
$$\quad \kappa(\eta[\![\mathbf{var}]\!][\![I]\!])$$

$$E[\![E_1\ \Omega\ E_2]\!]\eta\kappa =$$
$$\quad REval(\langle [\![E_1]\!], [\![E_2]\!]\rangle)\,\eta(\lambda\varepsilon^*.\kappa(W[\![\Omega]\!]\varepsilon^*))$$

$$E[\![\Theta \text{ before } E_1]\!]\eta\kappa =$$
$$\quad C[\![\Theta]\!]\eta\{E[\![E_1]\!]\eta\kappa\}$$

$$E[\![E_1 \rightarrow E_2, E_3]\!]\eta\kappa =$$
$$\quad R[\![E_1]\!]\eta\{\lambda\varepsilon.\varepsilon \rightarrow E[\![E_2]\!]\eta\kappa, E[\![E_3]\!]\eta\kappa\}$$

$$E[\![\text{let } \Delta \text{ in } E_1]\!]\eta\kappa =$$
$$\quad ClaimProcs(\eta[\![\text{proc}]\!])2\{\lambda\pi^*.D[\![\Delta]\!]\eta[\pi_1/\text{proc}]$$
$$\qquad \{\lambda\varepsilon^*.E[\![E_1]\!]\eta[\varepsilon^*/\text{var}/I[\![\Delta]\!]][\pi_2/\text{proc}]\kappa\}\}$$

$$D[\![I = E]\!]\eta\kappa' =$$
$$\quad R[\![E]\!]\eta\{\lambda\varepsilon.NewLoc(\eta[\![\text{proc}]\!])\{\lambda\alpha.Update\alpha\varepsilon\{\kappa'\langle\alpha\rangle\}\}\}$$

$$D[\![I \equiv E]\!]\eta\kappa' =$$
$$\quad R[\![E]\!]\eta\{\lambda\varepsilon.\kappa'\langle\varepsilon\rangle\}$$

$$D[\![\Delta_1 \text{ and } \Delta_2]\!]\eta\kappa' =$$
$$\quad DEval\langle D[\![\Delta_1]\!], D[\![\Delta_2]\!]\rangle\eta\{\lambda\langle\varepsilon_1^*, \varepsilon_2^*\rangle.\kappa'(\varepsilon_1^* \hat{} \varepsilon_2^*)\}$$

$$I[\![I = E]\!] = \langle I\rangle$$

$$I[\![I \equiv E]\!] = \langle I\rangle$$

$$I[\![\Delta_1 \text{ and } \Delta_2]\!] = I[\![\Delta_1]\!] \hat{} I[\![\Delta_2]\!]$$

$$L[\![E]\!]\eta\kappa = E[\![E]\!]\eta(L\upsilon\kappa)$$

$$R[\![E]\!]\eta\kappa = E[\![E]\!]\eta(R\nu\kappa)$$

$$L\nu\kappa\varepsilon = (\varepsilon\in L \rightarrow \kappa\varepsilon,\ Error)$$

$$R\nu\kappa\varepsilon = (\varepsilon\in L \rightarrow Contse\kappa,\ \varepsilon\in V \rightarrow \kappa\varepsilon,\ Error)$$

$$Updateo\alpha\beta\theta\sigma = \langle Put\alpha\beta, \theta\rangle$$

$$Contse\alpha\kappa\sigma = \langle CheckFetch\alpha(Fetch\alpha\sigma), \kappa(Fetch\alpha\sigma)\rangle$$

$$CheckFetch\alpha\beta\sigma = (\beta = Fetch\alpha\sigma \rightarrow \sigma,\ \underline{?})$$

$$Put\alpha\beta\sigma = \sigma[\beta/\mathrm{store}/\alpha]$$

$$Fetch\alpha\sigma = \sigma[\![\mathrm{store}]\!]\alpha$$

$$Break\theta\sigma = \langle \lambda\sigma.\sigma, \theta\rangle$$

$$NewLoc\pi\kappa\sigma = StoreFull\pi\sigma \rightarrow \langle CheckFull\pi, \theta_t\rangle,\ \langle CheckNew\pi(New\pi\sigma), \kappa(New\pi\sigma)\rangle$$

$$CheckFull\pi\sigma = (StoreFull\pi\sigma \rightarrow SetError\sigma,\ \underline{?})$$

$$CheckNew\pi\alpha\sigma = (StoreFull\pi\sigma \rightarrow \underline{?},\ \alpha = New\pi\sigma \rightarrow Use\alpha\sigma,\ \underline{?})$$

$$Use\alpha\sigma = \sigma[True/\mathrm{area}/\alpha]$$

Syntactic Equations:

$IdeVal[\![B]\!] = \langle\rangle$

$IdeVal[\![I]\!] = \langle I \rangle$

$IdeVal[\![E_1 \ \Omega \ E_2]\!] = \langle\rangle$

$IdeVal[\![\Theta \text{ before } E_1]\!] = IdeVal[\![E_1]\!]$

$IdeVal[\![E_1 \rightarrow E_2, E_3]\!] = IdeVal[\![E_2]\!] \ \hat{} \ IdeVal[\![E_3]\!]$

$IdeVal[\![\text{let } \Delta \text{ in } E_1]\!] = IdeVal[\![E_1]\!] \setminus I[\![\Delta]\!]$

$IdeL[\![E_1 := E_2]\!] = IdeL[\![E_1]\!] \ \hat{} \ IdeVal[\![E_1]\!] \ \hat{} \ IdeL[\![E_2]\!]$

$IdeL[\![\S I \ \Theta^* \ \S I]\!] = \bigwedge_{\iota=1}^{\#\Theta^*} IdeL[\![\Theta_\iota]\!]$

$IdeL[\![\text{while } E \text{ do } \Theta_1]\!] - IdeL[\![E]\!] \ \hat{} \ IdeL[\![\Theta_1]\!]$

$IdeL[\![\text{if } E \text{ do } \Theta_1]\!] = IdeL[\![E]\!] \ \hat{} \ IdeL[\![\Theta_1]\!]$

$IdeL[\![\text{escape } I]\!] = \langle\rangle$

$IdeL[\![\text{let } \Delta \text{ in } \Theta_1]\!] = IdeL[\![\Delta]\!] \ \hat{} \ (IdeL[\![\Theta_1]\!] \setminus I[\![\Delta]\!])$

$IdeL[\![B]\!] = \langle\rangle$

$IdeL[\![I]\!] = \langle\rangle$

$IdeL[\![E_1 \ \Omega \ E_2]\!] = IdeL[\![E_1]\!] \ \hat{} \ IdeL[\![E_2]\!]$

$IdeL[\![\Theta \text{ before } E_1]\!] = IdeL[\![\Theta]\!] \ \hat{} \ IdeL[\![E_1]\!]$

$IdeL[\![E_1 \rightarrow E_2, E_3]\!] = IdeL[\![E_1]\!] \ \hat{} \ IdeL[\![E_2]\!] \ \hat{} \ IdeL[\![E_3]\!]$

$IdeL[\![\text{let } \Delta \text{ in } E_1]\!] = IdeL[\![\Delta]\!] \ \hat{} \ (IdeL[\![E_1]\!] \setminus I[\![\Delta]\!])$

$IdeL[\![I = E]\!] = IdeL[\![E]\!]$

$IdeL[\![I \equiv E]\!] = IdeL[\![E]\!]$

$IdeL[\![\Delta_1 \text{ and } \Delta_2]\!] = IdeL[\![\Delta_1]\!] \ \hat{} \ IdeL[\![\Delta_2]\!]$

$IdeR[\![E_1 := E_2]\!] = IdeR[\![E_1]\!] \ \hat{} \ RIdeR[\![E_2]\!]$

$IdeR[\![\S I \ \Theta^* \ \S I]\!] = \bigwedge_{\iota=1}^{\#\Theta^*} IdeR[\![\Theta_\iota]\!]$

$IdeR[\![\text{while } E \text{ do } \Theta_1]\!] = RIdeR[\![E]\!] \ \hat{} \ IdeR[\![\Theta_1]\!]$

$IdeR[\![\text{if } E \text{ do } \Theta_1]\!] - RIdeR[\![E]\!] \ \hat{} \ IdeR[\![\Theta_1]\!]$

$IdeR[\![\text{escape } I]\!] = \langle\rangle$

$IdeR[\![\text{let } \Delta \text{ in } \Theta_1]\!] = IdeR[\![\Delta]\!] \ \hat{} \ (IdeR[\![\Theta_1]\!] \setminus I[\![\Delta]\!])$

$$IdeR[\![B]\!] = \langle\rangle$$

$$IdeR[\![I]\!] = \langle\rangle$$

$$IdeR[\![E_1 \ \Omega \ E_2]\!] = RIdeR[\![E_1]\!] \ \hat{} \ RIdeR[\![E_2]\!]$$

$$IdeR[\![\Theta \text{ before } E_1]\!] = IdeR[\![\Theta]\!] \ \hat{} \ IdeR[\![E_1]\!]$$

$$IdeR[\![E_1 \rightarrow E_2, E_3]\!] = RIdeR[\![E_1]\!] \ \hat{} \ IdeR[\![E_2]\!] \ \hat{} \ IdeR[\![E_3]\!]$$

$$IdeR[\![\text{let } \Delta \text{ in } E_1]\!] = IdeR[\![\Delta]\!] \ \hat{} \ (IdeR[\![E_1]\!] \setminus I[\![\Delta]\!])$$

$$IdeR[\![I = E]\!] = RIdeR[\![E]\!]$$

$$IdeR[\![I \equiv E]\!] = RIdeR[\![E]\!]$$

$$IdeR[\![\Delta_1 \text{ and } \Delta_2]\!] = IdeR[\![\Delta_1]\!] \ \hat{} \ IdeR[\![\Delta_2]\!]$$

$$RIdeR[\![E]\!] = IdeR[\![E]\!] \ \hat{} \ IdeVal[\![E]\!]$$

$$Jumps[\![E_1 := E_2]\!] = Jumps[\![E_1]\!] \ \hat{} \ Jumps[\![E_2]\!]$$

$$Jumps[\![\S I \ \Theta^* \ \S I]\!] = \bigwedge_{\iota=1}^{\#\Theta^*} Jumps[\![\Theta_\iota]\!] \setminus \langle I \rangle$$

$$Jumps[\![\text{while } E \text{ do } \Theta_1]\!] = Jumps[\![E]\!] \ \hat{} \ Jumps[\![\Theta_1]\!]$$

$$Jumps[\![\text{if } E \text{ do } \Theta_1]\!] = Jumps[\![E]\!] \ \hat{} \ Jumps[\![\Theta_1]\!]$$

$Jumps[\![\text{escape } I]\!] = \langle I \rangle$

$Jumps[\![\text{let } \Delta \text{ in } \Theta_1]\!] = Jumps[\![\Delta]\!] \hat{} Jumps[\![\Theta_1]\!]$

$Jumps[\![B]\!] = \langle\rangle$

$Jumps[\![I]\!] = \langle\rangle$

$Jumps[\![E_1 \ \Omega \ E_2]\!] = Jumps[\![E_1]\!] \hat{} Jumps[\![E_2]\!]$

$Jumps[\![\Theta \text{ before } E_1]\!] = Jumps[\![\Theta]\!] \hat{} Jumps[\![E_1]\!]$

$Jumps[\![E_1 \rightarrow E_2, E_3]\!] = Jumps[\![E_1]\!] \hat{} Jumps[\![E_2]\!] \hat{} Jumps[\![E_3]\!]$

$Jumps[\![\text{let } \Delta \text{ in } E_1]\!] = Jumps[\![\Delta]\!] \hat{} Jumps[\![E_1]\!]$

$Jumps[\![I = E]\!] = Jumps[\![E]\!]$

$Jumps[\![I \equiv E]\!] = Jumps[\![E]\!]$

$Jumps[\![\Delta_1 \text{ and } \Delta_2]\!] - Jumps[\![\Delta_1]\!] \hat{} Jumps[\![\Delta_2]\!]$

Appendix 2: Definition of POPLAR

"In its case it lay compactly,
Folded into nearly nothing;
But he opened out its hinges,
Pushed and pulled the joints and hinges
Till it looked all squares and oblongs,
Like a complicated figure
In the second book of Euclid."

Hiawatha's Photographing.

Syntactic Domains:

$B \in$ Bas

$\Theta \in$ Com ::= $E_1 := E_2$

§I Θ^* §I

while E do Θ_1

if E do Θ_1

escape I

let Δ in Θ_1

let Φ in Θ_1

$\Delta \in$ Dec ::= I = E

I ≡ E

Δ_1 and Δ_2

E $\in$ Exp ::= B

I

E_1 Ω E_2

Θ before E_1

$E_1 \rightarrow E_2, E_3$

let Δ in E_1

let Φ in E_1

$E_1[E^*]$

$\Phi \in$ Fun ::= $I[I^*;\Sigma^*] \equiv E$

$I \in Ide$

$\Omega \in Ope$

$\Sigma \in Spe ::= lv$

rv

Semantic Domains:

$\beta \in B$

$\theta \in C = [S \to [[S \to S] \times C]] + \{\theta_t\}$

$\delta \in D = L + B + F$

$\varepsilon \in E = L + B + F$

$\phi \in F = params:Ide^* \times spec:Spe^* \times body:Exp \times env:X \times gen:N$

$\gamma \in G = fn:[Ide \to N \to [\,]free:Ide^* \times rfree:Ide^* \times spe:Spe^* \times par:Ide^*]] \times$
$gen:N \times share:Ide^*$

$\eta \in H = proc:P \times var:[Ide \to D^*] \times cont:[Ide \to C] \times G$

$\iota \in I$

$\kappa \in K = E \to C$

$\kappa' \in K' = E^* \to C$

$\kappa'' \in K'' = E^{**} \to C$

$\alpha \in L$

$\nu \in N$

$\pi \in P$

$\sigma \in S = store:[L \to V] \times area:[L \to T] \times res:[P \to E] \times use:[P \to T] \times err:T$

$\tau \in T$

$V = B$

$\xi \in X = [Ide \to D^*]$

$\zeta \in Z = P^{\infty}$

$\chi \in [S \to S]$

A2

Semantics:

$$C[\![E_1 := E_2]\!]\eta\theta =$$
$$\quad LREval\langle [\![E_1]\!], [\![E_2]\!]\rangle\, \eta(\lambda\langle \alpha, \beta\rangle .\, Update\alpha\beta\theta)$$

$$C[\![\S I\ \Theta^*\ \$I]\!]\eta\theta =$$
$$\quad CEval[\![\Theta^*]\!](\eta[\theta/\mathsf{cont}/I])\theta$$

$$C[\![\mathbf{while}\ E\ \mathbf{do}\ \Theta_1]\!]\eta\theta =$$
$$\quad Fix(\lambda\theta'.R[\![E]\!]\eta\{\lambda\varepsilon.\varepsilon \rightarrow C[\![\Theta_1]\!]\eta(Break\theta'),\ \theta\})$$

$$C[\![\mathbf{if}\ E\ \mathbf{do}\ \Theta_1]\!]\eta\theta =$$
$$\quad R[\![E]\!]\eta\{\lambda\varepsilon.\varepsilon \rightarrow C[\![\Theta_1]\!]\eta\theta,\ \theta\}$$

$$C[\![\mathbf{escape}\ I]\!]\eta\theta =$$
$$\quad \eta[\![\mathsf{cont}]\!][\![I]\!]$$

$$C[\![\mathbf{let}\ \Delta\ \mathbf{in}\ \Theta_1]\!]\eta\theta =$$
$$\quad ClaimProcs(\eta[\![\mathsf{proc}]\!])2\{\lambda\pi^*.D[\![\Delta]\!]\eta[\pi_1/\mathsf{proc}]$$
$$\qquad \{\lambda\varepsilon^*.C[\![\Theta_1]\!](\eta[\varepsilon^*/\mathsf{var}/I[\![\Delta]\!]][\pi_2/\mathsf{proc}])\theta\}\}$$

$$C[\![\mathbf{let}\ \Phi\ \mathbf{in}\ \Theta_1]\!]\eta\theta$$
$$\quad F[\![\Phi]\!]\eta\{\lambda\phi^*.\ \ [\![\Theta_1]\!](NoteAbs[\![\Phi]\!](\eta[\phi^*\hat{}/\mathsf{var}/I[\![\Phi]\!]])(\eta[\![\mathsf{gen}]\!]+1))\theta\}$$

$$E[\![B]\!]\eta\kappa =$$
$$\quad \kappa(B[\![B]\!])$$

$$E[\![I]\!]\eta\kappa =$$
$$\quad \kappa(\eta[\![\mathsf{var}]\!][\![I]\!]\downarrow 1)$$

$$E[\![E_1\ \Omega\ E_2]\!]\eta\kappa =$$
$$\quad REval\langle [\![E_1]\!], [\![E_2]\!]\rangle\ \eta(\lambda\varepsilon^*.\kappa(W[\![\Omega]\!]\varepsilon^*))$$

$$E[\![\Theta \textbf{ before } E_1]\!]\eta\kappa =$$
$$\quad C[\![\Theta]\!]\eta\{E[\![E_1]\!]\eta\kappa\}$$

$$E[\![E_1 \rightarrow E_2, E_3]\!]\eta\kappa =$$
$$\quad R[\![E_1]\!]\eta\{\lambda\varepsilon.\varepsilon \rightarrow E[\![E_2]\!]\eta\kappa, E[\![E_3]\!]\eta\kappa\}$$

$$E[\![\textbf{let } \Delta \textbf{ in } E_1]\!]\eta\kappa =$$
$$\quad ClaimProcs(\eta[\![\textbf{proc}]\!])2\{\lambda\pi^*.D[\![\Delta]\!]\eta[\pi_1/\textbf{proc}]$$
$$\qquad \{\lambda\varepsilon^*.E[\![E_1]\!]\eta[\varepsilon^*/\textbf{var}/I[\![\Delta]\!]][\pi_2/\textbf{proc}]\kappa\}\}$$

$$E[\![\textbf{let } \Phi \textbf{ in } E_1]\!]\eta\kappa =$$
$$\quad F[\![\Phi]\!]\eta\{\lambda\phi^*.LR[\![E_1]\!](NoteAbs[\![\Phi]\!](\eta[\phi^*\hat{}/\textbf{var}/I[\![\Phi]\!]])(\eta[\![\textbf{gen}]\!]+1))\kappa\}$$

$$E[\![E_1[E^*]]\!]\eta\kappa =$$
$$\quad A[\![E_1]\!]\eta\{\lambda\phi.\#E^* \neq \#\phi[\![\textbf{params}]\!] \rightarrow Error,$$
$$\qquad ClaimProcs(\eta[\![\textbf{proc}]\!])2\{\lambda\pi^*.SEval(\phi[\![\textbf{spec}]\!])[\![E^*]\!]\eta[\pi_1/\textbf{proc}]$$
$$\qquad\quad \{\lambda\delta^*.ApplyFn[\![E_1]\!]\eta[\pi_2/\textbf{proc}]\phi\delta^*\kappa\}\}\}$$

$$D[\![I = E]\!]\eta\kappa' =$$
$$\quad R[\![E]\!]\eta\{\lambda\varepsilon.NewLoc(\eta[\![\textbf{proc}]\!])\{\lambda\alpha.Update\alpha\varepsilon\{\kappa'\langle\alpha\rangle\}\}\}$$

$$D[\![I \equiv E]\!]\eta\kappa' =$$
$$\quad R[\![E]\!]\eta\{\lambda\varepsilon.\kappa'\langle\varepsilon\rangle\}$$

A2

$$D[\![\Delta_1 \text{ and } \Delta_2]\!]\eta\kappa' =$$
$$DEval\langle D[\![\Delta_1]\!], D[\![\Delta_2]\!]\rangle\,\eta\{\lambda\langle \varepsilon_1^*, \varepsilon_2^*\rangle .\kappa'(\varepsilon_1^* \hat{} \varepsilon_2^*)\}$$

$$F[\![\Phi]\!]\eta\kappa' = \kappa'(Fix(\lambda\phi^*.G[\![\Phi]\!](\eta[\phi^* \hat{} /\text{var}/I[\![\Phi]\!]])(\eta[\![\text{gen}]\!]+1)))$$

$$G[\![\text{I}[\,\text{I}^*;\Sigma^*]\,]\!] \equiv \text{E}]\!]\eta\nu = \langle\langle \text{I}^*, \Sigma^*, \text{E}, \eta[\![\text{var}]\!], \nu\rangle\rangle$$

$$I[\![\text{I} = \text{E}]\!] = \langle \text{I}\rangle$$

$$I[\![\text{I} \equiv \text{E}]\!] = \langle \text{I}\rangle$$

$$I[\![\Delta_1 \text{ and } \Delta_2]\!] = I[\![\Delta_1]\!] \hat{} I[\![\Delta_2]\!]$$

$$L[\![\text{E}]\!]\eta\kappa = E[\![\text{E}]\!]\eta(Lv\kappa)$$

$$R[\![\text{E}]\!]\eta\kappa = E[\![\text{E}]\!]\eta(Rv\kappa)$$

$$LR[\![\text{E}]\!]\eta\kappa - E[\![\text{E}]\!]\eta\{\lambda\varepsilon.\varepsilon\epsilon\text{L}\vee\varepsilon\epsilon\text{V} \rightarrow \kappa\varepsilon,\ Error\}$$

$$A[\![\text{E}]\!]\eta\kappa = E[\![\text{E}]\!]\eta\{\lambda\varepsilon.\varepsilon\epsilon\text{F} \rightarrow \kappa\varepsilon,\ Error\}$$

$$Lv\kappa\varepsilon = (\varepsilon\epsilon\text{L} \rightarrow \kappa\varepsilon,\ Error)$$

$$Rv\kappa\varepsilon = (\varepsilon\epsilon\text{L} \rightarrow Conts\varepsilon\kappa,\ \varepsilon\epsilon\text{V} \rightarrow \kappa\varepsilon,\ Error)$$

$$Update\alpha\beta\theta\sigma = \langle Put\alpha\beta, \theta\rangle$$

$$Contsa\kappa\sigma = \langle CheckFetcha\alpha(Fetcha\alpha\sigma), \kappa(Fetcha\alpha\sigma)\rangle$$

$$CheckFetcha\alpha\beta\sigma = (\beta = Fetcha\alpha\sigma \rightarrow \sigma, \underline{?})$$

$$Puta\alpha\beta\sigma = \sigma[\beta/store/\alpha]$$

$$Fetcha\alpha\sigma = \sigma[\![store]\!]\alpha$$

$$Breaka\theta\sigma = \langle \lambda\sigma.\sigma, \theta\rangle$$

$$NewLoc\pi\kappa\sigma = StoreFull\pi\sigma \rightarrow \langle CheckFull\pi, \theta_t\rangle, \langle CheckNew\pi(New\pi\sigma), \kappa(New\pi\sigma)\rangle$$

$$CheckFull\pi\sigma = (StoreFull\pi\sigma \rightarrow SetError\sigma, \underline{?})$$

$$CheckNew\pi\alpha\sigma = (StoreFull\pi\sigma \rightarrow \underline{?}, \alpha = New\pi\sigma \rightarrow Use\alpha\sigma, \underline{?})$$

$$Use\alpha\sigma = \sigma[True/area/\alpha]$$

$$NoteAbs[\![I[I^*;\Sigma^*] \equiv E]\!]\eta\nu =$$
$$\eta[IdeL[\![E]\!]\gamma\nu/fn/I/\nu/lfree]$$
$$[RIdeR[\![E]\!]\gamma\nu/fn/I/\nu/rfree]$$
$$[\Sigma^*/fn/I/\nu/spe]$$
$$[I^*/fn/I/\nu/par]$$
$$[\nu/gen]$$

$$ApplyFn[\![E]\!]\eta\langle I^*,\Sigma^*,E',\xi,\nu\rangle\,\delta^*\kappa =$$

$$Break(R[\![E']\!](NoteAppn[\![E]\!](\eta[\xi[\delta^*\hat{}/I^*]/var][\lambda I.Error/cont]))\kappa)$$

$$NoteAppn[\![E]\!]\eta =$$

$$\eta[ShareList(AIdeVal[\![E]\!])\gamma/share]$$

$$ShareList I^*\gamma =$$

$$\bigwedge_{I\leftarrow I^*} \bigwedge_{\nu=1}^{\gamma[\![gen]\!]} (\lambda\psi.LParams(\psi[\![par]\!])(\psi[\![spe]\!]))\ (\gamma[\![fn]\!][\![I]\!]\nu)$$

$$LParams I^*\Sigma^* =$$

$$\bigwedge_{\iota=1}^{\#\Sigma^*} (\Sigma_\iota = 1\nu \rightarrow \langle I_\iota\rangle, \langle\rangle)$$

Syntactic Equations:

$IdeVal[\![B]\!] = \langle\rangle$

$IdeVal[\![I]\!] = \langle I \rangle$

$IdeVal[\![E_1 \ \Omega \ E_2]\!] = \langle\rangle$

$IdeVal[\![\Theta \text{ before } E_1]\!] = IdeVal[\![E_1]\!]$

$IdeVal[\![E_1 \rightarrow E_2, E_3]\!] = IdeVal[\![E_2]\!] \ \hat{} \ IdeVal[\![E_3]\!]$

$IdeVal[\![\text{let } \Delta \text{ in } E_1]\!] = IdeVal[\![E_1]\!] \setminus I[\![\Delta]\!]$

$IdeVal[\![\text{let } \Phi \text{ in } E_1]\!] = IdeVal[\![E_1]\!]$

$IdeVal[\![E_1[E^*]]\!] = \langle\rangle$

$AIdeVal[\![B]\!] = \langle\rangle$

$AIdeVal[\![I]\!] = \langle I \rangle$

$AIdeVal[\![E_1 \ \Omega \ E_2]\!] = \langle\rangle$

$AIdeVal[\![\Theta \text{ before } E_1]\!] = AIdeVal[\![E_1]\!]$

$AIdeVal[\![E_1 \rightarrow E_2, E_3]\!] = AIdeVal[\![E_2]\!] \ \hat{} \ AIdeVal[\![E_3]\!]$

$AIdeVal[\![\text{let } \Delta \text{ in } E_1]\!] = AIdeVal[\![E_1]\!] \setminus I[\![\Delta]\!]$

$$AIdeVal[\![\text{let } \Phi \text{ in } E_1]\!] = \langle\rangle$$

$$AIdeVal[\![E_1[E^*]]\!] = \langle\rangle$$

$$FreeLVal[\![E]\!]\gamma\nu = \bigwedge_{I \triangleleft AIdeVal[\![E]\!]} \bigwedge_{\iota=1}^{\nu} \gamma[\![\text{fn}]\!][\![I]\!]\iota[\![\text{lfree}]\!]$$

$$FreeRVal[\![E]\!]\gamma\nu = \bigwedge_{I \triangleleft AIdeVal[\![E]\!]} \bigwedge_{\iota=1}^{\nu} \gamma[\![\text{fn}]\!][\![I]\!]\iota[\![\text{rfree}]\!]$$

$$SpecVal[\![E]\!]\gamma\nu = \bigwedge_{I \triangleleft AIdeVal[\![E]\!]} \bigwedge_{\iota=1}^{\nu} \gamma[\![\text{fn}]\!][\![I]\!]\iota[\![\text{spe}]\!]$$

$$IdeL[\![E_1 := E_2]\!]\gamma\nu = IdeL[\![E_1]\!]\gamma\nu \hat{} IdeVal[\![E_1]\!] \hat{} IdeL[\![E_2]\!]\gamma\nu$$

$$IdeL[\![\S I\ \Theta^*\ \S I]\!]\gamma\nu = \bigwedge_{\iota=1}^{\#\Theta^*} IdeL[\![\Theta_\iota]\!]\gamma\nu$$

$$IdeL[\![\text{while } E \text{ do } \Theta_1]\!]\gamma\nu = IdeL[\![E]\!]\gamma\nu \hat{} IdeL[\![\Theta_1]\!]\gamma\nu$$

$$IdeL[\![\text{if } E \text{ do } \Theta_1]\!]\gamma\nu = IdeL[\![E]\!]\gamma\nu \hat{} IdeL[\![\Theta_1]\!]\gamma\nu$$

$$IdeL[\![\text{escape } I]\!]\gamma\nu = \langle\rangle$$

$$IdeL[\![\text{let } \Delta \text{ in } \Theta_1]\!]\gamma\nu = IdeL[\![\Delta]\!]\gamma\nu \hat{} IdeL[\![\Theta_1]\!]\gamma\nu$$

$$IdeL[\![\text{let } \Phi \text{ in } \Theta_1]\!]\gamma\nu = IdeL[\![\Phi]\!]\gamma\nu \hat{} IdeL[\![\Theta_1]\!]\gamma\nu$$

$$IdeL[\![B]\!]\gamma\nu = \langle\rangle$$

$$IdeL[\![I]\!]\gamma\nu = \langle\rangle$$

$$IdeL[\![E_1 \ \Omega \ E_2]\!]\gamma\nu = IdeL[\![E_1]\!]\gamma\nu \ \hat{} \ IdeL[\![E_2]\!]\gamma\nu$$

$$IdeL[\![\Theta \textbf{ before } E_1]\!]\gamma\nu = IdeL[\![\Theta]\!]\gamma\nu \ \hat{} \ IdeL[\![E_1]\!]\gamma\nu$$

$$IdeL[\![E_1 \rightarrow E_2, E_3]\!]\gamma\nu = IdeL[\![E_1]\!]\gamma\nu \ \hat{} \ IdeL[\![E_2]\!]\gamma\nu \ \hat{} \ IdeL[\![E_3]\!]\gamma\nu$$

$$IdeL[\![\textbf{let } \Delta \textbf{ in } E_1]\!]\gamma\nu = IdeL[\![\Delta]\!]\gamma\nu \ \hat{} \ IdeL[\![E_1]\!]\gamma\nu$$

$$IdeL[\![\textbf{let } \Phi \textbf{ in } E_1]\!]\gamma\nu = IdeL[\![\Phi]\!]\gamma\nu \ \hat{} \ IdeL[\![E_1]\!]\gamma\nu$$

$$IdeL[\![E_1[E^*]]\!]\gamma\nu =$$

$$IdeL[\![E_1]\!]\gamma\nu \ \hat{} \ \bigwedge_{\iota=1}^{\#E^*} (IdeL[\![E_\iota]\!]\gamma\nu \ \hat{} \ IdeVal[\![E_\iota]\!]) \ \hat{} \ FreeLVal[\![E_1]\!]\gamma\nu$$

$$IdeL[\![I = E]\!]\gamma\nu = IdeL[\![E]\!]\gamma\nu$$

$$IdeL[\![I \equiv E]\!]\gamma\nu = IdeL[\![E]\!]\gamma\nu$$

$$IdeL[\![\Delta_1 \textbf{ and } \Delta_2]\!]\gamma\nu = IdeL[\![\Delta_1]\!]\gamma\nu \ \hat{} \ IdeL[\![\Delta_2]\!]\gamma\nu$$

$$IdeL[\![I[I^*;\Sigma^*] \equiv E]\!]\gamma\nu = IdeL[\![E]\!]\gamma\nu$$

$$IdeR[\![E_1 := E_2]\!]\gamma\nu = IdeR[\![E_1]\!]\gamma\nu \ \hat{} \ RIdeR[\![E_2]\!]\gamma\nu$$

$$IdeR[\![\S I\ \Theta^*\ \$I]\!]\gamma\nu = \bigwedge_{\iota=1}^{\#\Theta^*} IdeR[\![\Theta_\iota]\!]\gamma\nu$$

$$IdeR[\![\textbf{while}\ E\ \textbf{do}\ \Theta_1]\!]\gamma\nu = RIdeR[\![E]\!]\gamma\nu \hat{}\ IdeR[\![\Theta_1]\!]\gamma\nu$$

$$IdeR[\![\textbf{if}\ E\ \textbf{do}\ \Theta_1]\!]\gamma\nu = RIdeR[\![E]\!]\gamma\nu \hat{}\ IdeR[\![\Theta_1]\!]\gamma\nu$$

$$IdeR[\![\textbf{escape}\ I]\!]\gamma\nu = \langle\rangle$$

$$IdeR[\![\textbf{let}\ \Delta\ \textbf{in}\ \Theta_1]\!]\gamma\nu - IdeR[\![\Delta]\!]\gamma\nu \hat{}\ IdeR[\![\Theta_1]\!]\gamma\nu$$

$$IdeR[\![\textbf{let}\ \Phi\ \textbf{in}\ \Theta_1]\!]\gamma\nu - IdeR[\![\Phi]\!]\gamma\nu \hat{}\ IdeR[\![\Theta_1]\!]\gamma\nu$$

$$IdeR[\![B]\!]\gamma\nu = \langle\rangle$$

$$IdeR[\![I]\!]\gamma\nu = \langle\rangle$$

$$IdeR[\![E_1\ \Omega\ E_2]\!]\gamma\nu = RIdeR[\![E_1]\!]\gamma\nu \hat{}\ RIdeR[\![E_2]\!]\gamma\nu$$

$$IdeR[\![\Theta\ \textbf{before}\ E_1]\!]\gamma\nu = IdeR[\![\Theta]\!]\gamma\nu \hat{}\ IdeR[\![E_1]\!]\gamma\nu$$

$$IdeR[\![E_1 \rightarrow E_2, E_3]\!]\gamma\nu = RIdeR[\![E_1]\!]\gamma\nu \hat{}\ IdeR[\![E_2]\!]\gamma\nu \hat{}\ IdeR[\![E_3]\!]\gamma\nu$$

$$IdeR[\![\textbf{let}\ \Delta\ \textbf{in}\ E_1]\!]\gamma\nu - IdeR[\![\Delta]\!]\gamma\nu \hat{}\ IdeR[\![E_1]\!]\gamma\nu$$

$$IdeR[\![\textbf{let}\ \Phi\ \textbf{in}\ E_1]\!]\gamma\nu = IdeR[\![\Phi]\!]\gamma\nu \hat{}\ IdeR[\![E_1]\!]\gamma\nu$$

$$IdeR[\![E_1[E^*]]\!]\gamma\nu =$$

$$IdeR[\![E_1]\!]\gamma\nu \hat{}\ \bigwedge_{\iota=1}^{\#E^*} RIdeR[\![E_\iota]\!]\gamma\nu \hat{}\ FreeRVal[\![E_1]\!]\gamma\nu$$

$$IdeR[\![I = E]\!]\gamma\nu = RIdeR[\![E]\!]\gamma\nu$$

$$IdeR[\![I \equiv E]\!]\gamma\nu = RIdeR[\![E]\!]\gamma\nu$$

$$IdeR[\![\Delta_1 \text{ and } \Delta_2]\!]\gamma\nu = IdeR[\![\Delta_1]\!]\gamma\nu \hat{} IdeR[\![\Delta_2]\!]\gamma\nu$$

$$IdeR[\![I[I^*;\Sigma^*] \equiv E]\!]\gamma\nu = IdeR[\![E]\!]\gamma\nu$$

$$RIdeR[\![E]\!]\gamma\nu = IdeR[\![E]\!]\gamma\nu \hat{} IdeVal[\![E]\!]$$

$$Jumps[\![E_1 := E_2]\!] = Jumps[\![E_1]\!] \hat{} Jumps[\![E_2]\!]$$

$$Jumps[\![\S I\ \Theta^*\ \S I]\!] = \bigwedge_{\iota=1}^{\#\Theta^*} Jumps[\![\Theta_\iota]\!] \setminus \langle I \rangle$$

$$Jumps[\![\textbf{while } E \textbf{ do } \Theta_1]\!] = Jumps[\![E]\!] \hat{} Jumps[\![\Theta_1]\!]$$

$$Jumps[\![\textbf{if } E \textbf{ do } \Theta_1]\!] = Jumps[\![E]\!] \hat{} Jumps[\![\Theta_1]\!]$$

$$Jumps[\![\textbf{escape } I]\!] = \langle I \rangle$$

$$Jumps[\![\textbf{let } \Delta \textbf{ in } \Theta_1]\!] = Jumps[\![\Delta]\!] \hat{} Jumps[\![\Theta_1]\!]$$

$$Jumps[\![\textbf{let } \Phi \textbf{ in } \Theta_1]\!] = Jumps[\![\Theta_1]\!]$$

$$Jumps[\![B]\!] = \langle\rangle$$

$Jumps[\![I]\!] = \langle\rangle$

$Jumps[\![E_1 \ \Omega \ E_2]\!] = Jumps[\![E_1]\!] \ \hat{} \ Jumps[\![E_2]\!]$

$Jumps[\![\Theta \textbf{ before } E_1]\!] = Jumps[\![\Theta]\!] \ \hat{} \ Jumps[\![E_1]\!]$

$Jumps[\![E_1 \rightarrow E_2, E_3]\!] = Jumps[\![E_1]\!] \ \hat{} \ Jumps[\![E_2]\!] \ \hat{} \ Jumps[\![E_3]\!]$

$Jumps[\![\textbf{let } \Delta \textbf{ in } E_1]\!] = Jumps[\![\Delta]\!] \ \hat{} \ Jumps[\![E_1]\!]$

$Jumps[\![\textbf{let } \Phi \textbf{ in } E_1]\!] = Jumps[\![E_1]\!]$

$$Jumps[\![E_1[E^*]]\!] = Jumps[\![E_1]\!] \ \hat{} \ \bigwedge_{\iota=1}^{\#E^*} Jumps[\![E_\iota]\!]$$

$Jumps[\![I = E]\!] = Jumps[\![E]\!]$

$Jumps[\![I \equiv E]\!] = Jumps[\![E]\!]$

$Jumps[\![\Delta_1 \textbf{ and } \Delta_2]\!] = Jumps[\![\Delta_1]\!] \ \hat{} \ Jumps[\![\Delta_2]\!]$

Appendix 3: Definition of APPLE

"Dreaming of apples on a wall,
And dreaming often, dear."

Puzzles from Wonderland.

Syntactic Domains:

$B \in$ Bas

$\Theta \in$ Com ::= $E_1 := E_2$

§I Θ^* §I

while E do Θ_1

if E do Θ_1

escape I

let Δ in Θ_1

for I from E_1 to E_2 do Θ_1

$\Delta \in$ Dec ::= I = E

I ≡ E

I ≡ array E^*

Δ_1 and Δ_2

E $\in$ Exp ::= B

I

E_1 Ω E_2

Θ before E_1

$E_1 \rightarrow E_2, E_3$

let Δ in E_1

I↓E^*

A3

$I \in \mathbf{Ide}$

$N \in \mathbf{Nml}$

$\Omega \in \mathbf{Ope} ::= +$

$\times$

Semantic Domains:

$\beta \in B$

$\theta \in C = [S \to [[S \to S] \times C]] + \{\theta_t\}$

$\delta \in D = L + B + R$

$\varepsilon \in E = L + B + R$

$\eta \in H = proc:P \times var:[Ide \to D] \times cont:[Ide \to C] \times lit:[Ide \to T]$

$\iota \in I$

$J = [Ide \times [I^* + \{?\}]^*] + Ide$

$\kappa \in K = E \to C$

$\kappa' \in K' = E^* \to C$

$\kappa'' \in K'' = E^{**} \to C$

$\alpha \in L$

$\nu \in N$

$\pi \in P$

$\rho \in R = bounds:I^* \times locs:L^*$

$\sigma \in S = store:[L \to V] \times area:[L \to T] \times res:[P \to E] \times use:[P \to T] \times err:T$

$\tau \in T$

$V = B$

$\zeta \in Z = P^\infty$

$\chi \in [S \to S]$

Semantics:

$$C[\![E_1 := E_2]\!]\eta\theta =$$
$$\quad LREval\langle [\![E_1]\!], [\![E_2]\!]\rangle\, \eta(\lambda\langle \alpha, \beta\rangle .\, Update\alpha\beta\theta)$$

$$C[\![\S I\ \Theta^*\ \S I]\!]\eta\theta =$$
$$\quad CEval[\![\Theta^*]\!](\eta[\theta/\mathrm{cont}/I])\theta$$

$$C[\![\mathbf{while}\ E\ \mathbf{do}\ \Theta_1]\!]\eta\theta =$$
$$\quad Fix(\lambda\theta'.R[\![E]\!]\eta\{\lambda\varepsilon.\varepsilon \rightarrow C[\![\Theta_1]\!]\eta(Break\theta'),\ \theta\})$$

$$C[\![\mathbf{if}\ E\ \mathbf{do}\ \Theta_1]\!]\eta\theta =$$
$$\quad R[\![E]\!]\eta\{\lambda\varepsilon.\varepsilon \rightarrow C[\![\Theta_1]\!]\eta\theta,\ \theta\}$$

$$C[\![\mathbf{escape}\ I]\!]\eta\theta =$$
$$\quad \eta[\![\mathrm{cont}]\!][\![I]\!]$$

$$C[\![\mathbf{let}\ \Delta\ \mathbf{in}\ \Theta_1]\!]\eta\theta =$$
$$\quad ClaimProcs(\eta[\![\mathrm{proc}]\!])2\{\lambda\pi^*.D[\![\Delta]\!]\eta[\pi_1/\mathrm{proc}]$$
$$\qquad \{\lambda\varepsilon^*.C[\![\Theta_1]\!]\eta[\varepsilon^*/\mathrm{var}/I[\![\Delta]\!]][NewLit[\![\Delta]\!]/\mathrm{lit}/I[\![\Delta]\!]][\pi_2/\mathrm{proc}]\theta\}\}$$

$$C[\![\mathbf{for}\ I\ \mathbf{from}\ E_1\ \mathbf{to}\ E_2\ \mathbf{do}\ \Theta_1]\!]\eta\theta =$$
$$\quad REval\langle [\![E_1]\!], [\![E_2]\!]\rangle\, \eta\{\lambda\langle \varepsilon_1, \varepsilon_2\rangle .\, QEval\langle \varepsilon_1, \varepsilon_2\rangle [\![I]\!][\![\Theta_1]\!]\eta\theta\}$$

$$Q[\![I]\!]\iota[\![\Theta]\!]\eta =$$
$$\quad C[\![\Theta]\!]\eta[\iota/\mathrm{var}/I][True/\mathrm{lit}/I]$$

$$E[\![B]\!]\eta\kappa =$$
$$\quad \kappa(B[\![B]\!])$$

$$E[\![I]\!]\eta\kappa =$$
$$\quad \kappa(\eta[\![\mathbf{var}]\!][\![I]\!])$$

$$E[\![E_1 \ \Omega \ E_2]\!]\eta\kappa =$$
$$\quad \mathit{REval}\langle [\![E_1]\!], [\![E_2]\!] \rangle \eta(\lambda\varepsilon^*.\kappa(\mathit{W}[\![\Omega]\!]\varepsilon^*))$$

$$E[\![\Theta \ \mathbf{before} \ E_1]\!]\eta\kappa =$$
$$\quad C[\![\Theta]\!]\eta\{E[\![E_1]\!]\eta\kappa\}$$

$$E[\![E_1 \rightarrow E_2, \ E_3]\!]\eta\kappa =$$
$$\quad R[\![E_1]\!]\eta\{\lambda\varepsilon.\varepsilon \rightarrow E[\![E_2]\!]\eta\kappa, \ E[\![E_3]\!]\eta\kappa\}$$

$$E[\![\mathbf{let} \ \Delta \ \mathbf{in} \ E_1]\!]\eta\kappa =$$
$$\quad \mathit{ClaimProcs}(\eta[\![\mathbf{proc}]\!])2\{\lambda\pi^*.D[\![\Delta]\!]\eta[\pi_1/\mathbf{proc}]$$
$$\qquad \{\lambda\varepsilon^*.E[\![E_1]\!]\eta[\varepsilon^*/\mathbf{var}/I[\![\Delta]\!]][\mathit{NewLit}[\![\Delta]\!]/\mathbf{lit}/I[\![\Delta]\!]][\pi_2/\mathbf{proc}]\kappa\}\}$$

$$E[\![I{\downarrow}E^*]\!]\eta\kappa =$$
$$\quad (\lambda\varepsilon.\varepsilon\notin R\rightarrow ?,$$
$$\qquad \mathit{REval}[\![E^*]\!]\eta\{\lambda\varepsilon^*.\mathit{InArrayRange}\varepsilon\varepsilon^*\rightarrow\kappa(\mathit{GetArrayLoc}\varepsilon\varepsilon^*), ?\})$$
$$\quad (\eta[\![\mathbf{var}]\!][\![I]\!])$$

$$D[\![I = E]\!]\eta\kappa' =$$
$$\quad R[\![E]\!]\eta\{\lambda\varepsilon.\mathit{NewLoc}(\eta[\![\mathbf{proc}]\!])\{\lambda\alpha.\mathit{Update}\alpha\varepsilon\{\kappa'\langle\alpha\rangle\}\}\}$$

$$D[\![I \equiv E]\!]\eta\kappa' =$$
$$\quad R[\![E]\!]\eta\{\lambda\varepsilon.\kappa'\langle\varepsilon\rangle\}$$

$$D[\![I \equiv \textbf{array } E^*]\!]\eta\kappa' =$$
$$\quad REval[\![E^*]\!]\eta\{\lambda\varepsilon^*.NewLocs(\eta[\![\textbf{proc}]\!])(\Pi\varepsilon^*)(\lambda\alpha^*.\kappa'(\langle \varepsilon^*, \alpha^* \rangle))\}$$

$$D[\![\Delta_1 \textbf{ and } \Delta_2]\!]\eta\kappa' =$$
$$\quad DEval\langle D[\![\Delta_1]\!], D[\![\Delta_2]\!]\rangle \eta\{\lambda\langle \varepsilon_1^*, \varepsilon_2^* \rangle .\kappa'(\varepsilon_1^* \hat{} \varepsilon_2^*)\}$$

$$W[\![+]\!]\langle \varepsilon_1, \varepsilon_2 \rangle = \varepsilon_1 + \varepsilon_2$$

$$W[\![\times]\!]\langle \varepsilon_1, \varepsilon_2 \rangle = \varepsilon_1 \times \varepsilon_2$$

$$I[\![I = E]\!] = \langle I \rangle$$

$$I[\![I \equiv E]\!] = \langle I \rangle$$

$$I[\![I \equiv \textbf{array } E^*]\!] = \langle I \rangle$$

$$I[\![\Delta_1 \textbf{ and } \Delta_2]\!] = I[\![\Delta_1]\!] \hat{} I[\![\Delta_2]\!]$$

$$L[\![E]\!]\eta\kappa = E[\![E]\!]\eta(Lv\kappa)$$

$$R[\![E]\!]\eta\kappa = E[\![E]\!]\eta(Rv\kappa)$$

$$NewLit[\![I = E]\!] = \langle True \rangle$$

$$NewLit[\![I \equiv E]\!] = \langle False \rangle$$

$$NewLit[\![I \equiv \text{array } E^*]\!] = \langle False \rangle$$

$$NewLit[\![\Delta_1 \text{ and } \Delta_2]\!] = NewLit[\![\Delta_1]\!] \hat{} NewLit[\![\Delta_2]\!]$$

$$Lv\kappa\varepsilon = (\varepsilon \in L \rightarrow \kappa\varepsilon, Error)$$

$$Rv\kappa\varepsilon = (\varepsilon \in L \rightarrow Contse\kappa, \varepsilon \in V \rightarrow \kappa\varepsilon, Error)$$

$$Update\alpha\beta\theta\sigma = \langle Put\alpha\beta, \theta \rangle$$

$$Contsa\kappa\sigma = \langle CheckFetch\alpha(Fetch\alpha\sigma), \kappa(Fetch\alpha\sigma) \rangle$$

$$CheckFetch\alpha\beta\sigma = (\beta = Fetch\alpha\sigma \rightarrow \sigma, \underline{?})$$

$$Put\alpha\beta\sigma = \sigma[\beta/\text{store}/\alpha]$$

$$Fetch\alpha\sigma = \sigma[\![\text{store}]\!]\alpha$$

$$Break\theta\sigma = \langle \lambda\sigma.\sigma, \theta \rangle$$

$$NewLoc\pi\kappa\sigma = StoreFull\pi\sigma \rightarrow \langle CheckFull\pi, \theta_t \rangle ,$$
$$\langle CheckNew\pi(New\pi\sigma), \kappa(New\pi\sigma) \rangle$$

$$CheckFull\pi\sigma = (StoreFull\pi\sigma \rightarrow SetError\sigma, \underline{?})$$

$$CheckNew\pi\alpha\sigma = (StoreFull\pi\sigma \rightarrow \underline{?},$$
$$\alpha = New\pi\sigma \rightarrow Use\alpha\sigma, \underline{?})$$

$$Use\alpha\sigma = \sigma[True/\text{area}/\alpha]$$

Syntactic Equations:

$$IdeVal[\![B]\!][\![I^*]\!] = \langle\rangle$$

$$IdeVal[\![I]\!][\![I^*]\!] = \langle I \rangle$$

$$IdeVal[\![E_1 \ \Omega \ E_2]\!][\![I^*]\!] = \langle\rangle$$

$$IdeVal[\![\Theta \text{ before } E_1]\!][\![I^*]\!] = IdeVal[\![E_1]\!][\![I^*]\!]$$

$$IdeVal[\![E_1 \rightarrow E_2, E_3]\!][\![I^*]\!] = IdeVal[\![E_2]\!][\![I^*]\!] \ \hat{} \ IdeVal[\![E_3]\!][\![I^*]\!]$$

$$IdeVal[\![\text{let } \Delta \text{ in } E_1]\!][\![I^*]\!] = IdeVal[\![E_1]\!](Purge[\![I^*]\!][\![\Delta]\!]) \setminus I[\![\Delta]\!]$$

$$IdeVal[\![I \downarrow E^*]\!][\![I^*]\!] = \langle\langle I, \mathop{*}_{\iota=1}^{\#E^*} IdeSub[\![E_\iota]\!][\![I^*]\!] \rangle\rangle$$

$$IdeSub[\![I]\!][\![I^*]\!] = (I \lessdot I^* \rightarrow \mathop{*}_{\iota=1}^{\#I^*+1} (I = I_\iota \rightarrow 1, 0), ?)$$

$$IdeSub[\![N]\!][\![I^*]\!] = (\mathop{*}_{\iota=1}^{\#I^*} (0)) \hat{} \langle B[\![N]\!] \rangle$$

$$IdeSub[\![N \times I]\!][\![I^*]\!] =$$

$$IdeSub[\![I \times N]\!][\![I^*]\!] = (I \lessdot I^* \rightarrow \mathop{*}_{\iota=1}^{\#I^*+1} (I = I_\iota \rightarrow B[\![N]\!], 0), ?)$$

$$IdeSub[\![E_1 + E_2]\!][\![I^*]\!] =$$

$$\left(\underset{\iota=1}{\overset{\#I^*+1}{*}} (IdeSub[\![E_1]\!][\![I^*]\!]\downarrow\iota + IdeSub[\![E_2]\!][\![I^*]\!]\downarrow\iota)\right)$$

for all other cases:

$$IdeSub[\![E]\!][\![I^*]\!] = ?$$

$$Purge[\![I^*]\!][\![I \equiv E]\!] = I^*$$

$$Purge[\![I^*]\!][\![I = E]\!] =$$

$$Purge[\![I^*]\!][\![I \equiv \textbf{array } E^*]\!] = Nullify[\![I^*]\!][\![I]\!]$$

$$Purge[\![I^*]\!][\![\Delta_1 \textbf{ and } \Delta_2]\!] = Purge(Purge[\![I^*]\!][\![\Delta_1]\!])[\![\Delta_2]\!]$$

$$Nullify[\![I^*]\!][\![I]\!] =$$

$$(\#I^*=0 \rightarrow \langle\rangle\,,\ \langle I=I_1 \rightarrow I_0, I_1\rangle \hat{}\, Nullify(I^*\dagger 1)[\![I]\!])$$

$$IdeL[\![E_1 := E_2]\!][\![I^*]\!] =$$

$$IdeL[\![E_1]\!][\![I^*]\!] \hat{}\ IdeVal[\![E_1]\!][\![I^*]\!] \hat{}\ IdeL[\![E_2]\!][\![I^*]\!]$$

$$IdeL[\![\S I\ \Theta^*\ \S I]\!][\![I^*]\!] = \underset{\iota=1}{\overset{\#\Theta^*}{\Lambda}} IdeL[\![\Theta_\iota]\!][\![I^*]\!]$$

$$IdeL[\![\textbf{while } E \textbf{ do } \Theta_1]\!][\![I^*]\!] = IdeL[\![E]\!][\![I^*]\!] \hat{}\ IdeL[\![\Theta_1]\!][\![I^*]\!]$$

$$IdeL[\![\textbf{if } E \textbf{ do } \Theta_1]\!][\![I^*]\!] = IdeL[\![E]\!][\![I^*]\!] \hat{}\ IdeL[\![\Theta_1]\!][\![I^*]\!]$$

$$IdeL[\![\textbf{escape } I]\!][\![I^*]\!] = \langle\rangle$$

$$IdeL[\![\textbf{let } \Delta \textbf{ in } \Theta_1]\!][\![I^*]\!] = IdeL[\![\Delta]\!][\![I^*]\!] \hat{} (IdeL[\![\Theta_1]\!](Purge[\![I^*]\!][\![\Delta]\!]) \setminus I[\![\Delta]\!])$$

$$IdeL[\![\textbf{for } I \textbf{ from } E_1 \textbf{ to } E_2 \textbf{ do } \Theta_1]\!][\![I^*]\!] = IdeL[\![E_1]\!][\![I^*]\!] \hat{} IdeL[\![E_2]\!][\![I^*]\!] \hat{} IdeL[\![\Theta_1]\!][\![I^*]\!]$$

$$IdeL[\![B]\!][\![I^*]\!] = \langle\rangle$$

$$IdeL[\![I]\!][\![I^*]\!] = \langle\rangle$$

$$IdeL[\![E_1 \ \Omega \ E_2]\!][\![I^*]\!] = IdeL[\![E_1]\!][\![I^*]\!] \hat{} IdeL[\![E_2]\!][\![I^*]\!]$$

$$IdeL[\![\Theta \textbf{ before } E_1]\!][\![I^*]\!] = IdeL[\![\Theta]\!][\![I^*]\!] \hat{} IdeL[\![E_1]\!][\![I^*]\!]$$

$$IdeL[\![E_1 \rightarrow E_2, E_3]\!][\![I^*]\!] = IdeL[\![E_1]\!][\![I^*]\!] \hat{} IdeL[\![E_2]\!][\![I^*]\!] \hat{} IdeL[\![E_3]\!][\![I^*]\!]$$

$$IdeL[\![\textbf{let } \Delta \textbf{ in } E_1]\!][\![I^*]\!] = IdeL[\![\Delta]\!][\![I^*]\!] \hat{} (IdeL[\![E_1]\!](Purge[\![I^*]\!][\![\Delta]\!]) \setminus I[\![\Delta]\!])$$

$$IdeL[\![I{\downarrow}E^*]\!][\![I^*]\!] = \bigwedge_{\iota=1}^{\#E^*} IdeL[\![E_\iota]\!][\![I^*]\!]$$

$$IdeL[\![I = E]\!][\![I^*]\!] = IdeL[\![E]\!][\![I^*]\!]$$

$$IdeL[\![I \equiv E]\!][\![I^*]\!] = IdeL[\![E]\!][\![I^*]\!]$$

$$IdeL[\![\, I \equiv \textbf{array}\ E^* \,]\!][\![I^*]\!] = \bigwedge_{\iota=1}^{\#E^*} IdeL[\![E_\iota]\!][\![I^*]\!]$$

$$IdeL[\![\Delta_1 \ \textbf{and}\ \Delta_2]\!][\![I^*]\!] = IdeL[\![\Delta_1]\!][\![I^*]\!] \ \hat{}\ IdeL[\![\Delta_2]\!][\![I^*]\!]$$

$$IdeR[\![E_1 := E_2]\!][\![I^*]\!] = IdeR[\![E_1]\!][\![I^*]\!] \ \hat{}\ RIdeR[\![E_2]\!][\![I^*]\!]$$

$$IdeR[\![\, \S I\ \Theta^*\ \S I]\!][\![I^*]\!] = \bigwedge_{\iota=1}^{\#\Theta^*} IdeR[\![\Theta_\iota]\!][\![I^*]\!]$$

$$IdeR[\![\textbf{while}\ E\ \textbf{do}\ \Theta_1]\!][\![I^*]\!] = RIdeR[\![E]\!][\![I^*]\!] \ \hat{}\ IdeR[\![\Theta_1]\!][\![I^*]\!]$$

$$IdeR[\![\textbf{if}\ E\ \textbf{do}\ \Theta_1]\!][\![I^*]\!] = RIdeR[\![E]\!][\![I^*]\!] \ \hat{}\ IdeR[\![\Theta_1]\!][\![I^*]\!]$$

$$IdeR[\![\textbf{escape}\ I]\!][\![I^*]\!] = \langle\rangle$$

$$IdeR[\![\textbf{let}\ \Delta\ \textbf{in}\ \Theta_1]\!][\![I^*]\!] =$$
$$\quad IdeR[\![\Delta]\!][\![I^*]\!] \ \hat{}\ (IdeR[\![\Theta_1]\!](Purge[\![I^*]\!][\![\Delta]\!]) \setminus I[\![\Delta]\!])$$

$$IdeR[\![\textbf{for}\ I\ \textbf{from}\ E_1\ \textbf{to}\ E_2\ \textbf{do}\ \Theta_1]\!][\![I^*]\!] =$$
$$\quad RIdeR[\![E_1]\!][\![I^*]\!] \ \hat{}\ RIdeR[\![E_2]\!][\![I^*]\!] \ \hat{}\ IdeR[\![\Theta_1]\!][\![I^*]\!]$$

$$IdeR[\![B]\!][\![I^*]\!] = \langle\rangle$$

$$IdeR[\![I]\!][\![I^*]\!] = \langle\rangle$$

$$IdeR[\![E_1\ \Omega\ E_2]\!][\![I^*]\!] = RIdeR[\![E_1]\!][\![I^*]\!] \ \hat{}\ RIdeR[\![E_2]\!][\![I^*]\!]$$

$$IdeR[\![\Theta\ \textbf{before}\ E_1]\!][\![I^*]\!] = IdeR[\![\Theta]\!][\![I^*]\!] \ \hat{}\ IdeR[\![E_1]\!][\![I^*]\!]$$

$$IdeR[\![E_1 \rightarrow E_2, E_3]\!][\![I^*]\!] =$$
$$\quad RIdeR[\![E_1]\!][\![I^*]\!] \;\hat{}\; IdeR[\![E_2]\!][\![I^*]\!] \;\hat{}\; IdeR[\![E_3]\!][\![I^*]\!]$$

$$IdeR[\![\textbf{let } \Delta \textbf{ in } E_1]\!][\![I^*]\!] =$$
$$\quad IdeR[\![\Delta]\!][\![I^*]\!] \;\hat{}\; (IdeR[\![E_1]\!](Purge[\![I^*]\!][\![\Delta]\!]) \setminus I[\![\Delta]\!])$$

$$IdeR[\![I{\downarrow}E^*]\!][\![I^*]\!] = \bigwedge_{\iota=1}^{\#E^*} RIdeR[\![E_\iota]\!]$$

$$IdeR[\![I = E]\!][\![I^*]\!] = RIdeR[\![E]\!][\![I^*]\!]$$

$$IdeR[\![I \equiv E]\!][\![I^*]\!] = RIdeR[\![E]\!][\![I^*]\!]$$

$$IdeR[\![I \equiv \textbf{array } E^*]\!][\![I^*]\!] = \bigwedge_{\iota=1}^{\#E^*} RIdeR[\![E_\iota]\!][\![I^*]\!]$$

$$IdeR[\![\Delta_1 \textbf{ and } \Delta_2]\!][\![I^*]\!] = IdeR[\![\Delta_1]\!][\![I^*]\!] \;\hat{}\; IdeR[\![\Delta_2]\!][\![I^*]\!]$$

$$RIdeR[\![E]\!][\![I^*]\!] = IdeR[\![E]\!][\![I^*]\!] \;\hat{}\; IdeVal[\![E]\!][\![I^*]\!]$$

$$Jumps[\![E_1 := E_2]\!] = Jumps[\![E_1]\!] \;\hat{}\; Jumps[\![E_2]\!]$$

$$Jumps[\![\S I \; \Theta^* \; \S I]\!] = \bigwedge_{\iota=1}^{\#\Theta^*} Jumps[\![\Theta_\iota]\!] \setminus \langle I \rangle$$

$$Jumps[\![\textbf{while } E \textbf{ do } \Theta_1]\!] = Jumps[\![E]\!] \;\hat{}\; Jumps[\![\Theta_1]\!]$$

$$Jumps[\![\text{if } E \text{ do } \Theta_1]\!] = Jumps[\![E]\!] \hat{} Jumps[\![\Theta_1]\!]$$

$$Jumps[\![\text{escape } I]\!] = \langle I \rangle$$

$$Jumps[\![\text{let } \Delta \text{ in } \Theta_1]\!] = Jumps[\![\Delta]\!] \hat{} Jumps[\![\Theta_1]\!]$$

$$Jumps[\![\text{for } I \text{ from } E_1 \text{ to } E_2 \text{ do } \Theta_1]\!] = Jumps[\![E_1]\!] \hat{} Jumps[\![E_2]\!] \hat{} Jumps[\![\Theta_1]\!]$$

$$Jumps[\![B]\!] = \langle\rangle$$

$$Jumps[\![I]\!] = \langle\rangle$$

$$Jumps[\![E_1 \ \Omega \ E_2]\!] = Jumps[\![E_1]\!] \hat{} Jumps[\![E_2]\!]$$

$$Jumps[\![\Theta \text{ before } E_1]\!] = Jumps[\![\Theta]\!] \hat{} Jumps[\![E_1]\!]$$

$$Jumps[\![E_1 \rightarrow E_2, E_3]\!] = Jumps[\![E_1]\!] \hat{} Jumps[\![E_2]\!] \hat{} Jumps[\![E_3]\!]$$

$$Jumps[\![\text{let } \Delta \text{ in } E_1]\!] = Jumps[\![\Delta]\!] \hat{} Jumps[\![E_1]\!]$$

$$Jumps[\![I \downarrow E^*]\!] = \bigwedge_{\iota=1}^{\#E^*} Jumps[\![E_\iota]\!]$$

$$Jumps[\![I = E]\!] = Jumps[\![E]\!]$$

$$Jumps[\![I \equiv E]\!] = Jumps[\![E]\!]$$

$$Jumps[\![\mathrm{I} \equiv \mathbf{array}\ \mathrm{E}^*]\!] = \bigwedge_{\iota=1}^{\#\mathrm{E}^*} Jumps[\![\mathrm{E}_\iota]\!]$$

$$Jumps[\![\Delta_1 \ \mathbf{and}\ \Delta_2]\!] = Jumps[\![\Delta_1]\!] \ \hat{}\ Jumps[\![\Delta_2]\!]$$

Acknowledgements

"Thus grew the tale of Wonderland:
  Thus slowly, one by one,
Its quaint events were hammered out -
  And now the tale is done,
And home we steer, a merry crew,
  Beneath the setting sun."

Alice's Adventures in Wonderland: Introduction.

The research whose results are described herein was supported by a *Science Research Council* studentship. Thanks are due to those who supervised the work: the late Professor Christopher Strachey, whose were the original ideas from which this thesis germinated, but who, unfortunately, has not lived to see its completion; and Mr. Joe Stoy, who ably took his place. Thanks also to all of my colleagues at the *Programming Research Group*, who have provided a congenial environment for research, and especially to those who have often made useful comments, and to those who have contributed to the hardware and software involved in producing this thesis in its final form. Mention must be made, finally, of Rev.C.L.Dodgson, late of Christ Church, Oxford, who provided all the quotations.

BIBLIOGRAPHY

"'Who's been repeating all that hard stuff to you?'
'I read it in a book,' said Alice."
Through the Looking Glass: Chap 6.

Names of periodicals are abbreviated as follows:

CACM: Communications of A.C.M.

CS: Computing Surveys.

IEEET: Transactions of I.E.E.E.

IPL: Information Processing Letters.

JACM: Journal of A.C.M.

FJCC: A.F.I.P.S. Fall Joint Computer Conference Proceedings.

SJCC: A.F.I.P.S. Spring Joint Computer Conference Proceedings.

[1] T.L.Adam, K.M.Chandy, J.R.Dickson,
A Comparison of List Schedules for Parallel Processing Systems;
CACM 17.12 685-90.

[2] D.A.Adams,
A Model for Parallel Computations;
in [15].

[3] J.P.Anderson,
Program Structures for Parallel Processing;
CACM 8.12 786-8.

[4] J.L.Baer,
A Survey of Some Theoretical Aspects of Multiprocessing;
CS 5.1 31-80.

[5] J.L.Baer, E.C.Russell,
Preparation and Evaluation of Computer Programs for Parallel Processing Systems;
in [15].

[6] A.B.Barak,
On the Parallel Evaluation of Division-Free Arithmetic Expressions with Fan-in of Three;
IPL 5.1 18-9.

[7] J.C.Beatty,
An Axiomatic Approach to Code Optimization for Expressions;
JACM 19.4 613-40.

[8] A.J.Bernstein,
Analysis of Programs for Parallel Processing;
IEEET EC15.5 757-63.

[9] R.P.Brent,
The Parallel Evaluation of General Arithmetic Expressions;
JACM 21.2 201-6.

[10] D.D.Chamberlin,
The Single-Assignment Approach to Parallel Processing;
1971 FJCC 263-9.

[11] J.B.Dennis,
First Version of a Data Flow Procedure Language;
Proc. Symposium on Programming, University of Paris, 1974.

[12] D.P.Geller, G.M.Weinberg,
The Principle of Sufficient Reason;
SIGPLAN 10.3 34-8.

[13] M.J.Gonzalez, C.V.Ramamoorthy,
Program Suitability for Parallel Processing;
IEEET C20 647-54.

[14] M.J.Gonzalez, C.V.Ramamoorthy,
Recognition and Representation of Parallel Processable Streams in Computer Programs;
in [15].

[15] L.C.Hobbs, et al
Parallel Processor Systems, Technologies, and Applications;
Spartan books, NY, 1970.

[16] P.R.Kosinski,
A Data Flow Language for Operating Systems Programming;
SIGPLAN 8.9 89-94.

[17] D.J.Kuck, Y.Muraoka,
Bounds on the Parallel Evaluation of Arithmetic Expressions Using Associativity and Commutativity;
Acta Informatica 3.3 203-16.

[18] D.J.Kuck, Y.Muraoka, S.C.Chen,
On the Number of Operations Simultaneously Executable in FORTRAN-Like Programs and their Resulting Speedup;
IEEET C21.12 1293-310.

[19] D.J.Kuck, et al
Measurements of Parallelism in Ordinary FORTRAN Programs;
Computer 7.1 (1974) 37-46.

[20] L.Lamport,
The Parallel Execution of DO Loops;
CACM 17.2 83-93.

[21] M.Lehman,
A Survey of Problems and Preliminary Results Concerning Parallel Processing and Parallel Processes;
Proc. IEEE 54.12 1889-901.

[22] R.E.Milne,
The Formal Semantics of Computer Languages and their Implementations;
PhD, University of Cambridge, 1974.

[23] R.E.Milne, C.Strachey,
A Theory of Programming Language Semantics;
Chapman and Hall, 1976.

[24] Y.Muraoka,
Parallelism Exposure and Exploitation in Programs;
PhD, University of Illinois, 1971.

[25] A.Opler,
Procedure-Oriented Language Statements to Facilitate Parallel Processing;
CACM 8.5 306-307.

[26] G.D.Plotkin,
A Powerdomain Construction;
DAI Research Report No 3 (1975).

[27] C.V.Ramamoorthy, M.J.Gonzalez,
A Survey of Techniques for Recognizing Parallel Processable Streams in Computer Programs;
1969 FJCC 1-15.

[28] C.V.Ramamoorthy, M.J.Gonzalez,
Subexpression Ordering in the Execution of Arithmetic Expressions;
CACM 14.7 479-85.

[29] E.W.Reigel,
Parallelism Exposure and Exploitation;
in [15].

[30] P.B.Schneck,
Automatic Recognition of Vector and Parallel Operations in a Higher Level Language;
SIGPLAN 7.11 45-52.

[31] H.S.Stone,
One-Pass Compilation of Arithmetic Expressions for a Parallel Processor;
CACM 10.4 220-3.

[32] H.S.Stone,
Problems of Parallel Computation;
in [34].

[33] L.G.Tesler, H.G.Enea,
A Language Design for Concurrent Processes;
1968 SJCC 402-8.

[34] J.F.Traub,
Complexity of Sequential and Parallel Numerical Algorithms;
Academic Press, NY, 1973.

[35] G.M.Weinberg, D.P.Geller, T.W.S.Plum,
IF-THEN-ELSE Considered Harmful;
SIGPLAN 10.8 34-44.

[36] S.Winograd,
On the Parallel Evaluation of Certain Arithmetic Expressions;
JACM 22.4 477-92.

[37] N.Wirth,
A Note on 'Program Structures for Parallel Processing';
CACM 9.5 320-1.

INDEX

"Definitions: Index indicates the degree, or power, to which a particle is raised. It consists of two letters, placed to the right of the symbol representing the particle. Thus, 'A.A.' signifies the 0th degree; 'B.A.' the first degree; and so on, till we reach 'M.A.' the second degree (the intermediate letters indicating fractions of a degree)."

Dynamics of a Parti-cle: Chap 2.